VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Metodiky popisu při vývoji SW díla

Methodology for SW Development

2011 Bc. Kamil Uher

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 5. května 2011 . . . . . .

Velice rád bych poděkoval Ing. Svatoplukovi Štolfovi, Ph.D. za odborné konzultace, které mi pomohly při vypracování této diplomové práce.

Abstrakt

Práce se zabývá metodikami softwarových systémů. Jsou nalezeny a popsány jednotlivé metodiky, které jsou pak následně rozděleny do skupin dle charakteristických znaků a navzájem porovnány. Vybrané agilní metodiky jsou aplikovány na vybrané projekty společnosti Siemens. V poslední kapitole byly doporučeny metodiky, které byly následně aplikovány na případové studie.

Abstrakt

The work deals with methods of software systems. They are identified and described the various methodologies, which are subsequently divided into groups according to the characteristics and compared with each other. Selected agile methodologies are applied to selected projects from Siemens. In the last chapter of the recommended methodologies, which were then applied to case studies.

Klíčová slova

Agilní metodiky, Crystal Clear, Dynamic System Development Method, DSDM, Extrémní programování, XP, Implementace softwarového procesu, Prototypování, Rational Unified Process, RUP, Rigorózní metodiky, SCRUM, Spirálový model, Vodopádový model

Key words

Agile methodologies, Crystal Clerar, Dynamic Systém Development Method, DSDM, Extreme programming, XP, implementation of software process, Prototype, Rational Unified Process, RUP, Rigolous methodologies, SCRUM, spiral model, water-fall model

Seznam použitých symbolů a zkratek

AG	Aktiengesellschaft (akciová společnost)
Atd.	a tak dále
CMM	Capability
CC	Crystal Clear
DSDM	Dynamic System Development Method
ESS	Employee Self-Service
GSS	Global Shared Service
HR	Human Resources
IT	informační technologie
max.	maximálně
min.	minimálně
RUP	Rational Unified Process
SEI	Software Engineering Institute
SW	software
tzv.	tak zvaný
tzn.	to znamená
UML	Unfited Modeling Language
XP	Extrémní programování

Obsah

1 Úvod - 1 -
2 Nalezení dostupných a používaných metodik a jejich popis - 2 -
2.1 Vodopádový model - 3 -
2.1.1 Fáze vodopádového modelu - 3 -
2.1.2 Výhody a nevýhody vodopádového modelu - 4 -
2.2 Spirálový model - 5 -
2.2.1 Fáze spirálového modelu - 5 -
2.2.2 Výhody a nevýhody spirálového modelu - 7 -
2.3 Prototypování - 7 -
2.3.1 Fáze prototypovaného modelu - 7 -
2.3.2 Výhody a nevýhody prototypování - 8 -
2.4 Rational Unified Process (RUP) - 8 -
2.4.1 Obecné praktiky metodiky RUP - 8 -
2.4.2 Fáze RUP modelu - 9 -
2.4.3 Výhody a nevýhody RUP - 10 -
2.5 SCRUM - 11 -
2.5.1 Výhody a nevýhody - 12 -
2.6 Extrémní programování - 13 -
2.6.1 Životní cyklus - 14 -
2.6.2 Výhody a nevýhody - 15 -
2.7 Dynamic System Development Method - 15 -
2.7.1 Principy DSDM - 16 -
2.7.2 Životní cyklus - 16 -
2.7.3 Výhody a nevýhody - 18 -
2.8 Crystal Method - 18 -
2.8.1 Principy metodiky Crystal Clear (CC) - 19 -
2.8.2 Životní cyklus - 19 -
2.8.3 Výhody a nevýhody - 20 -
3 Roztřídění metodik do skupin dle charakteristických společných znaků - 22 -
3.1 Rigorózní metodiky - 23 -
3.1.1 Hodnocení rigorózních metodik - 23 -
3.2 Agilní metodiky - 24 -
3.2.1 Agilní principy - 25 -
3.2.2 Hodnocení agilních metodik - 26 -
3.3 Srovnání agilních metodik s rigorózními - 26 -
4 Porovnání metodik - 29 -
4.1 Porovnání vybraných agilních metodik podle principů - 29 -
4.1.1 Extrémní programování - 29 -
4.1.2 Dynamic System Development Method - 30 -
4.1.3 Crystal Method - 31 -
4.1.4 SCRUM - 31 -
4.1.5 Srovnání - 32 -
4.2 Porovnání agilních metodik podle rolí - 32 -
4.2.1 Extrémní programování - 32 -
4.2.2 Dynamic System Development Method - 33 -
4.2.3 Crystal Method - 34 -
4.2.4 SCRUM - 35 -

4.2.5 Srovnání ..... - 35 -
4.3 Porovnání agilních metodik podle postupů ..... - 36 -
4.3.1 Extrémní programování ..... - 36 -
4.3.2 Dynamic System Development Method ..... - 37 -
4.3.3 Crystal Method ..... - 38 -
4.3.4 SCRUM ..... - 39 -
4.3.5 Srovnání ..... - 40 -
5 Použití konkrétní metodiky na projektu ve firmě SIEMENS ..... - 41 -
5.1 Představení společnosti SIEMENS a popis projektu ESS portál ..... - 41 -
5.2 Role SCRUM při vývoji ESS portálu ..... - 44 -
5.3 Zahájení projektu a průběh při vývoji ESS portálu ..... - 45 -
5.4 Vyhodnocení metodiky SCRUM ..... - 46 -
5.5 Dokumentace projektu ..... - 48 -
6 Doporučení metodik pro modelové případy ..... - 49 -
6.1 Aplikace metodiky Extrémní programování na případové studii ..... - 49 -
6.1.1 Charakteristika případové studie ..... - 49 -
6.1.2 Extrémní programování ..... - 49 -
6.2 Aplikace metodiky DSDM na případové studii ..... - 52 -
6.2.1 Charakteristika případové studie ..... - 52 -
6.2.2 Dynamic System Development Method ..... - 53 -
6.3 Aplikace metodiky Cryslar Clear na případové studii ..... - 54 -
6.3.1 Charakteristika případové studie ..... - 54 -
6.3.2 Crystal Clear ..... - 54 -
7 Závěr ..... - 57 -

Seznam obrázků

Obrázek 1 – vodopádový model [14] ..... - 3 -
Obrázek 2 – spirálový model [15] ..... - 6 -
Obrázek 3 – životní cyklus RUP [16] ..... - 10 -
Obrázek 4 – SCRUM proces [17] ..... - 12 -
Obrázek 5 – životní cyklus DSDM [1] ..... - 18 -
Obrázek 6 – životní cyklus Crystal Method [8] ..... - 20 -
Obrázek 7 – ukázka ESS CZ ..... - 42 -
Obrázek 8 – ukázka ESS Cz Bankovního spojení ..... - 42 -
Obrázek 9 – ukázka ESS portálu, Dovolená ..... - 43 -
Obrázek 10 – ukázka ESS portálu, Přehled dovolených ..... - 43 -
Obrázek 11 – Scrum process [15] ..... - 46 -
Obrázek 12 – Ukázka rezervace zasedací místnosti ..... - 51 -
Obrázek 13 – Ukázka rezervovaných zasedacích místností ..... - 52 -
Obrázek 14 – Ukázka Sledování nákladů na zaměstnance ..... - 54 -
Obrázek 15 – Ukázka aplikace Docházka ..... - 56 -

Seznam tabulek

Tabulka 1 – srovnání podle principů ..... - 32 -
Tabulka 2 – srovnání podle rolí ..... - 35 -
Tabulka 3 – srovnání podle postupů ..... - 40 -

1 Úvod

S vývojem informačních technologií roste úměrně i potřeba vývoje software. V celé historii vývoje software se toto odvětví potýkalo s řadou problémů a vývojových změn. Přes „vodopádový model“, který stál na počátku celého vývoje software, přes „spirálový model“ se vývoj samotného software dostal na úroveň novějších, modernějších metodik zvaných Agilní metodiky, které se vyznačují větší rychlostí, kvalitou a nižšími finančními nároky. Agilní metodiky, které v dnešní době zažívají velký rozvoj, využívají nový přístup úzké a časté spolupráce se zákazníkem.

V první části diplomové práci se věnuji nalezení dostupných metodik a jejich popisu. Cílem této kapitoly je bližší seznámení s jednotlivými klasickými a agilními metodikami a jejich podrobnějšími popisu. Ve druhé části jsem roztřídil používané metodiky, které jsem blíže zmínil v první části, do skupin podle charakteristických společných znaků na metodiky rigorózní a metodiky agilní. Dále jsem se v této kapitole věnoval zhodnocení těchto metodik. Ve třetí kapitole jsem se věnoval porovnání jednotlivých metodik podle principů, rolí a jednotlivých postupů. Čtvrtá kapitola je věnována popisu použitých metodik na konkrétních projektech ve firmě Siemens. Jednotlivých metodik existuje velké množství, ale neznamená, že každá se hodí k nasazení v určité firmě. Proto je před nasazením zvážit, zda bude daná metodika firmě vyhovovat. V páté, závěrečné kapitole jsou popsány aplikace vybraných metodik na případových studiích ve firmě Siemens.

Cílem práce je získání přehledu o jednotlivých metodikách vývoje software a jejich porovnání. Druhým cílem je vyhodnocení jednotlivých metodik a použití vhodných metodik na vybrané konkrétní projekty a modelové případy ve společnosti Siemens.

2 Nalezení dostupných a používaných metodik a jejich popis

Metodika je jedním z nejčastěji opakujícím se pojmem v této práci, proto je nutné objasnit, co pojem metodika znamená v praxi.

Pojem metodika představuje v obecném chápání předem stanovený pracovní postup či návod, který nám stanovuje, jak postupovat při určitém úkonu.

V oblasti IT a ve vývoji software představuje metodika souhrn doporučených praktik a postupů, pokrývající celý životní cyklus vytváření aplikace.

Softwarový proces je po částech uspořádaná množina kroků směřujících k vytvoření nebo úpravě softwarového díla.

K výše zmíněné definici softwarového procesu se vztahují tyto výroky:

- krokem může být aktivita nebo podproces,
- aktivity a podprocesy mohou probíhat v čase souběžně, je tudíž nutná jejich koordinace,
- cílem je dosáhnout stabilních výsledků vysoké úrovně kvality, proto je nutné zajistit opakovatelnost,
- použití procesu ve vztahu k jednotlivým softwarovým projektům a zajistit jeho znovupoužitelnost,
- činnosti jsou zajišťovány lidmi disponujícími určitými schopnostmi a znalostmi a mají k dispozici technické prostředky nezbytné k realizaci těchto činností,
- softwarový produkt je realizován v kontextu organizace s danými ekonomickými možnostmi a organizační strukturou organizace.

Využití softwarového procesu je hodnocena podle stupnice **SEI** (Software Engineering Institute), která stupnicí bodů 1-5 vyjadřuje vyspělost firmy či organizace z daného hlediska. Tento model hodnocení vyspělosti a schopnosti dodavatele se nazývá Capability Maturity Model (CMM).

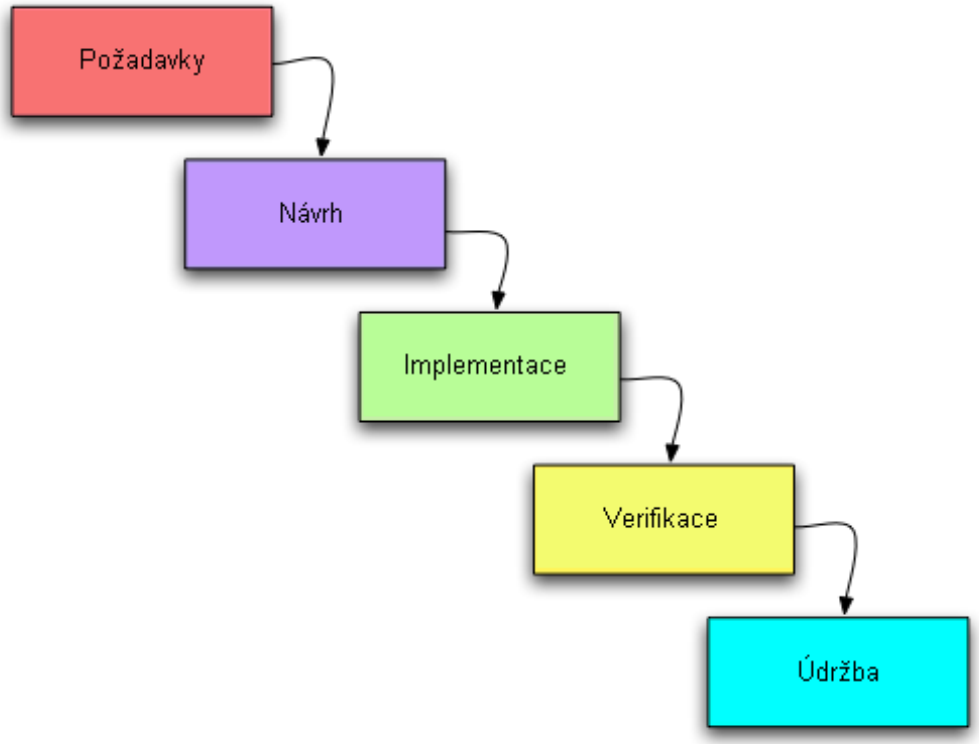
Úrovně modelu CMM jsou popsány níže:

1. Počáteční (Intial) – firma nemá definován softwarový proces a každý projekt je řešen případ od případu (ad hoc).
2. Opakovatelná (Repetable) – firma rozpoznala v jednotlivých projektech opakovatelné postupy a je schopna je využít v každém novém projektu.

3. Definovaná (Defined) – softwarový proces je definován a dokumentován na základě integrace dříve identifikovaných opakovatelných kroků.
4. Řízená (Managed) – na základě definovaného softwarového procesu je firma schopna jeho řízení a monitorování.
5. Optimalizovaná (Optimalized) – zpětnovazební informace získaná dlouhodobým procesem monitorování softwarového procesu je využita ve prospěch optimalizace.

2.1 Vodopádový model

Vodopádový model je základním modelem softwarového procesu, který se v jeho modifikacích nachází ve většině současných procesů. Jeho počátky nalezneme ve stavebnickém průmyslu a výrobě, kde bylo třeba nadefinovat a specifikovat výrobní či stavební postup. Postupem času se tento model začal rozšiřovat také v softwarovém inženýrství, kde byli jakékoli změny zadání, řešení a návrhu velmi nákladné. První kdo tento model popsal a specifikoval, i když tento termín ve své práci nepoužil, byl Dr. Winston W. Royce v roce 1970 v publikaci Proceeding institutu IEEE.



Obrázek 1 – vodopádový model [14]

2.1.1 Fáze vodopádového modelu

Vodopádový model vychází z rozdělení životního cyklu softwarového díla na čtyři základní fáze:

1. Analýza požadavků a jejich specifikace.

2. Návrh softwarového systému a specifikace požadavků software
3. Implementace (kódování).
4. Testování a udržování vytvořeného produktu.

Fáze sou sekvenčně seřazeny a navazují na sebe ve výše zmíněném pořadí, přičemž princip vodopádu spočívá v tom, že následující množina činností spjatá s danou fází nemůže započít dříve, než když skončí fáze předcházející. V tomto modelu je možné pohybovat se o jednu fázi vpřed nebo jednu fázi vzad.

Tato tradiční metodika má však i svoje nedostatky, jako jsou:

- dlouhá prodleva mezi zadáním projektu a vytvořením spustitelného sytému,
- výsledek závisí na úplném a korektním zadání požadavků na výsledný produkt,
- není možné odhadnout výslednou kvalitu produktu, dokud není hotov výsledný softwarový systém.

Snahou odstranit tyto nedostatky vznikly jednotlivé modifikace základního vodopádového modelu jako je model inkrementální postavený na postupném vytváření celé řady menších vodopádů s výrazně kratším životním cyklem, kde každý z nich odpovídá nové sadě doplňujících požadavků.

2.1.2 Výhody a nevýhody vodopádového modelu

Výhody:

- jednoduchost,
- snadné řízení dle metodiky,
- snadná dokumentace,
- výhodné pro velmi malé projekty,
- snadná kontrola stádia postupu projektu,
- není rušen zákazník v průběhu vývoje,
- snadná analýza dopadu požadované změny.

Nevýhody:

- neflexibilní,
- vyšší náklady při změně požadavků,
- složitá a objemná dokumentace,

- dražší metodika pro zákazníka,
- menší přímá komunikace se zákazníkem,
- zákazník vidí až konečný produkt,
- nutnost všechny požadavky definovat předem,
- problémy nastanou až ve fázi testování,
- chyba v prvotních fázích znamená krach projektu.

2.2 Spirálový model

Spirálový model byl navržen v roce 1985 Barry Boehmem a je znán také pod názvem Boehm´s Model. Vycházel z několika tehdejších modelů, které postupem času přestávaly rostoucí složitostí a zvětšující m se rozsahem softwarových projektů postačovat. Vznikl jako reakce na nedostatek vodopádového modelu a to na nemožnost reakce na změny požadavků.

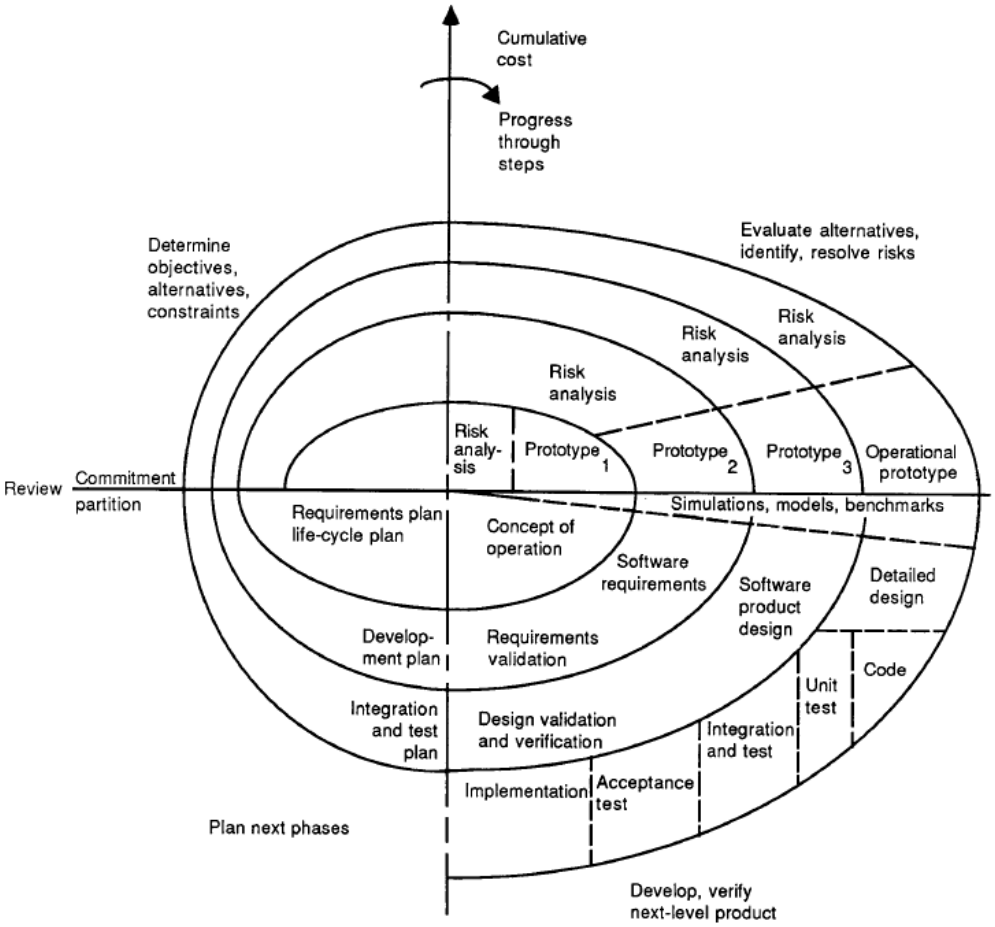
Spirálový model se řadí do skupiny tzv. riziky řízených přístupů a znamená to, že další postup v jednotlivých fázích závisí především na opakovaně a důsledně prováděné analýze všech rizik a možných problémů. Ve spirálovém modelu je obsažen model vodopádový. Rozdíl spočívá v tom, že postup fází není lineární, jako je tomu u modelu vodopádového, ale spirálově se točí s pravidelným odhalováním a analýzou rizik s vytvářením prototypů.

Spirálový model spočívá v silnějším rozdělení životního cyklu projektu do vyhodnotitelných kroků, což umožňuje zřetelně snižovat riziko výskytu chyb a tím náklady na jejich odstraňování zejména ve velkých a dlouhodobých projektech. Tento typ modelu není vhodný pro malé projekty, protože v každé fázi vyžaduje vyšší náklady na určování cílů, evaluaci, vývoj a následné plánování.

2.2.1 Fáze spirálového modelu

Spirálový model je rozdělen do 4 fází:

1. Analýza cílů, alternativ a omezení.
2. Analýza rizik, vyhodnocení alternativ.
3. Vývoj aplikace.
4. Plánování další fáze.



Obrázek 2 – spirálový model [15]

První fáze cyklu zahrnuje rozpoznání cílů, alternativ implementace a veškerá omezení aplikace. Druhá fáze spočívá ve vyhodnocení nejvhodnějších alternativ, které vyhovují zadaným cílům a omezením z první fáze. Ve třetí fázi se vyhodnotí výše rizika neúspěchu a podle něj se přizpůsobí další vývoj aplikace. Při skutečnosti, že se jedná o riziko malé, můžeme přistoupit k implementaci vodopádového modelu. Pokud je riziko příliš velké, přistoupí se nejprve ke zpracování požadavků na software. Pokud není systém kompletně hotov, přistoupí se ke čtvrté fázi. Čtvrtá fáze plánuje, na základě poznatků z předchozích fází, průběh dalších fází.

Spirálový model znamená, že čím větší je rádius spirály, tím větší je cena nebo čas pro vytvoření aplikace. Znamená to, že aplikace s menším rizikem neúspěchu jsou levnější, než

ty s rizikem vyšším.

Spirálový model se v dnešní době díky jeho náročnosti na řízení a nepružnosti reakcí na změny při implementaci nepoužívá. Jeho uplatnění je možné jen u velkých projektů, u kterých jsou jasně definované požadavky.

2.2.2 Výhody a nevýhody spirálového modelu

Výhody:

- iterativní přístup,
- nezávislost k metodice,
- analýza rizik v průběhu vývoje,
- vývojový tým se dostane dříve ke své práci.

Nevýhody:

- nevhodné při implementaci na malé projekty,
- jedná se o poměrně složitý model,
- je potřeba zkušeného týmu,
- projekt je zákazníkovi předán až po ukončení,
- vyšší cena spojená s vysokou režií,
- nepružný model při změně požadavků.

2.3 Prototypování

Proces prototypování vznikl v 70. letech, kdy reagoval na vytváření kompletního programu najednou a poté postupnou úpravu dle požadavků a připomínek, což vedlo ke špatným odhadům časů, zdrojů a hlavně k vyšší ceně aplikace. Pokud zákazník řekl, že je řešení špatné, muselo se začít od začátku. Jedná se o klasickou metodu, ale některé z novodobých metodik z ní přímo vycházejí a lze ji u některých projektů použít beze změn.

Prototypování je spíše procesem v rámci nějaké metodiky, která má za úkol předvést nejen zákazníkovi co možná nejvěrnější pohled na návrh systému. Tento pohled je zajištěn prototypem aplikace, která se může pod odzkoušení celá kompletně „zahodit“ nebo se může prototyp interativně rozšiřovat dále. Pokud dojde k dalšímu rozšiřování prototypu, můžeme ji považovat za metodiku jako takovou. Prototypování je vlastně proces rychlého vytvoření modelu finálního systému, který je primárně využit jako komunikační nástroj ke zpřístupnění informací a seznámení se s potřebami uživatele.

2.3.1 Fáze prototypovaného modelu

Fáze se dělí do pěti fází:

1. Sepsání prvotních požadavků.
2. Vývoj prvotního prototypu.

3. Vyhodnocení prototypu.
4. Revize a případné zlepšení prototypu.
5. Implementace finální podoby systému na základě prototypů.

2.3.2 Výhody a nevýhody prototypování

Výhody:

- lepší řízení změn a levnější a rychleji vytvořené dílo,
- možnost ostrého spuštění částečně funkčního systému,
- možnost lepšího odhadu času a zdrojů na základě prototypů,
- prototyp je prezentován zákazníkovi, který se k němu může poměrně brzy vyjádřit.

Nevýhody:

- žádná nebo jen základní analýza celého projektu,
- složitost údržby výsledné směsi prototypů,
- prototyp může vzbuzovat dojem, že je aplikace téměř hotová.

2.4 Rational Unified Process (RUP)

Kořeny této metodiky sahají na začátek 80. let 20. Století a jsou spjaty s firmou Rational Software. Jejími zakladateli jsou Paul Levy a Mike Devlin. Tato firma si kladla za úkol úspěšně vyvíjet velké komplexní softwarové systémy.

RUP je komerční metodika společnosti IBM poskytující systematický přístup k vývoji SW. Obsahuje detailní návody jak postupovat, aby byly splněny cíle a očekávání zadavatelů zakázky a zajištění vytvoření produktu vysoké kvality požadované zákazníkem v rámci predikovaného rozpočtu a časového rozvrhu. Popisuje fáze životního cyklu vývoje SW, definuje potřebné role, artefakty, odpovědnosti a pracovní procesy.

2.4.1 Obecné praktiky metodiky RUP

Metodika RUP se na vývoj SW soustředila pomocí šesti základních obecných praktik

1. Iterativní vývoj SW

Je alternativou vodopádovému životnímu cyklu, která umožňuje lepší porozumění problému díky postupnému vylepšování vyvíjeného SW. Interativní vývoj lépe odhaluje vysoká rizika v každé fázi životního cyklu, a tak výrazně snižuje jejich dopady. Rizika je možné odhadnout v prvních fázích cyklu, a tak je možné je hned na počátku zcela eliminovat a zamezit tím napadení celého projektu.

2. Správa a řízení poplatků

Na počátku projektu je možné, že zákazník přesně neví, co od projektu očekávají a své požadavky mění a specifikují až v průběhu času. Proto je nutné, aby tyto požadavky byly následně spravovány. Tyto požadavky mají významný dopad na ekonomické a technické cíle systému a jejich identifikace představuje nepřetržitý proces.

3. Použití komponentové architektury

Architektura systému je nejdůležitější součástí, která má vliv na různé úhly pohledu a zároveň má dopad na iterativní a inkrementální vývoj systému během životního cyklu.

Použití komponentové architektury spolu s praktikou iterativního vývoje SW mají za následek neustálý rozvoj architektury systému.

4. Vizuální modelování SW

Modelování je důležitou součást systému, protože napomáhá vývojovému týmu zobrazit, specifikovat, tvořit a dokumentovat strukturu a chování architektury systému. Mezi hlavní využívané modely jsou modely případů užití, diagramy scénářů, diagramy tříd, stavové diagramy, komponentované diagramy a diagramy dodání.

5. Průběžné kontrolování kvality

Průběžné kontrolování kvality a hodnocení systému je důležité na kvalitu sytému s ohledem na jeho funkcionalitu, spolehlivost, aplikační a systémovou výkonnost. Kontrola funkcionality systému zahrnuje vytváření a provádění sady testů pro každý klíčový scénář.

Mezi hlavní přínosy patří hodnocení stavu projektu je objektivní na základě výsledku formálních testů a neprovádí se testy hodnocení subjektivní papírové podobě.

6. Řízení změn

Koordinace činností a artefaktů různých týmů vyžaduje vytvoření opakovaného pracovního procesu řídícího změny SW a artefaktů. Spolu s praktikou iterativního vývoje umožňuje průběžné monitorování změn, což vede k odhalení problémů, na něž je možné následně reagovat.

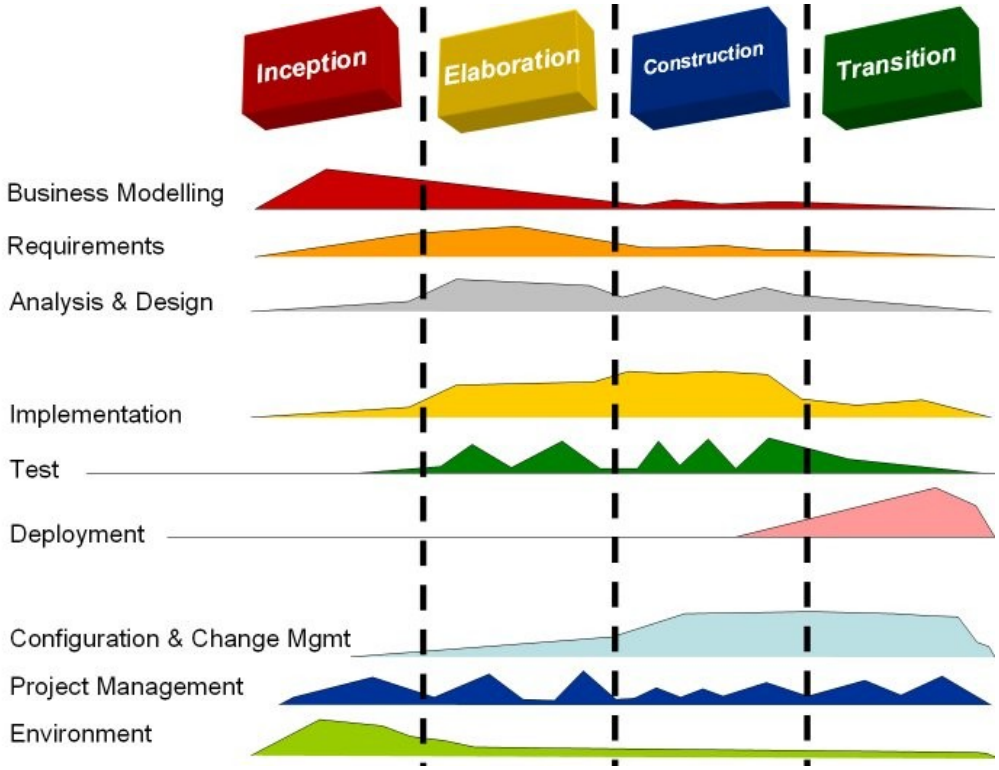
2.4.2 Fáze RUP modelu

Fáze životního cyklu se dělí na 4 fáze:

1. Zahájení – původní myšlenka je rozpracována do vize koncového produktu.
2. Rozpracování – podrobná specifikace požadavků a rozpracování architektury výsledného produktu.
3. Tvorba – kompletní vyhotovení požadovaného díla.

4. Předání – závěrečná fáze, kdy je vytvořený produkt předán do užívání.

Každá fáze se dělí do několika iterací, což jsou úkony, vedoucí k vytvoření spustitelné verze systému reprezentující podmnožinu vyvíjeného cílového produktu a která je postupně rozšiřována každou iterací až do výsledné podoby.



Obrázek 3 – životní cyklus RUP [16]

2.4.3 Výhody a nevýhody RUP

Výhody:

- brzké rozpoznání chyb,
- možnost brzké zpětné vazby zákazníka,
- soustředění týmu na konkrétní úkoly,
- iterativní testování umožňuje hodnocení projektu během vývoje,
- iterativní,
- jasně popsaná metodika,
- možnost implementace nekompletního díla.

Nevýhody:

- nemožnost reakce na změny požadavků,

- celý tým musí mít znalosti o UML syntax,
- nevhodné pro menší projekty,
- složitost a potřeba mít kvalitní zkušený tým.

2.5 SCRUM

Metodika SCRUM se netýká jenom vývoje software, ale lze ji použít i v jiných oblastech technologického vývoje. SCRUM je oproti jiným agilním metodikám poměrně striktně definován.

SCRUM se spoléhá na tým, který se řídí sám, bez dohledu managementu. Nemá definovaného formálního vedoucího týmu, ale spoléhá na profesionalitu jednotlivých členů týmu. Vývoj je rozdělen do etap, označovaných jako „sprinty“. Před začátkem každého sprintu se tým musí dohodnout na vlastnostech budovaného softwaru a soustředit se jen na ně. Výsledkem každého spritu je program, který je připravený k předvedení či předání zákazníkovi.

Role SCRUM

Scrum Master – osoba, která se nejvíce podobá vedoucímu týmu. Slouží jako prostředník mezi týmem a okolím. Dohlíží na dodržování postupů Scrumu, popřípadě odstraňuje vzniklé překážky a stará se o blaho týmu.

Product Owner – zákazník, určující které vlastnosti budou vyvíjeny během jednotlivých sprintů a stanovuje jejich priority. Tyto priority může během vývoje měnit.

Tým – tým SCRUM tvoří jednotlivý vývojáři. Optimální počet členů je mezi pěti a devíti, ale se stoupajícím počtem členů vznikají problémy v komunikaci mezi jednotlivými členy týmu.

Artefakty SCRUM

Procesy SCRUMU jsou podporovány třemi artefakty:

Product Backlog – jedná se o seznam všech požadavků na vyvíjený produkt, jeho vlastnosti a změny. Product Backlog je v průběhu vývoje průběžně doplňován a udržován.

Sprint Backlog – z Product Backlog jsou ze začátku každého sprintu vybrány nejdůležitější požadavky a z těch je tvořen Spring Backlog. Jednotlivé požadavky jsou rozděleny na menší úkoly, snadněji splnitelné. Jednotlivé úkony si vybírají sami vývojáři.

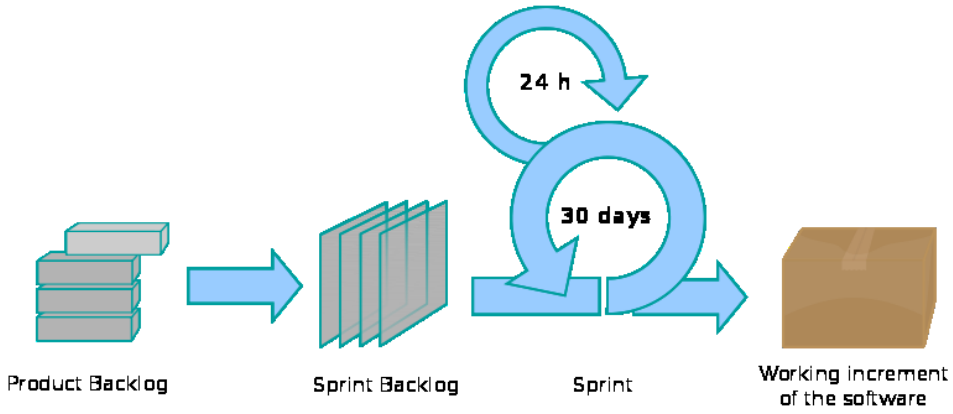
Burn Down Chart – výchozím znakem je Brun down graf, který zobrazuje počet dosud nesplněných úkolů v čase. Osa X zobrazuje datum, zatím co osa Y zobrazuje nesplněné úkoly. Graf by měl mít klesající lineární průběh.

Schůzky SCRUM

Daily Scrum je každodenní setkání všech členů týmu, probíhající ve stanovenou dobu. Doporučená doba trvání je 15 minut a Daily Scrum se může zúčastnit jako pozorovatel kdokoliv. Daily Scrum pomáhá týmu udržovat přehled o projektu a stavu vývoje.

Sprint Planning Meeting se koná vždy před začátkem každého sprintu, při němž se stanovuje jeho průběh a očekávaný výsledek. Setkání by mělo trvat max. čtyři hodiny.

Sprint Review Meeting se koná na konci každého sprintu a hodnotí to, co bylo během sprintu dokončeno a naopak. U tohoto setkání je vítána účast zákazníka a setkání by nemělo přesáhnout 4 hodiny. Cílem je zhodnotit podmínky pro práci, najít problematická místa a ta v příštím sprintu odstranit.



Obrázek 4 – SCRUM proces [17]

2.5.1 Výhody a nevýhody

Výhody:

- všechny kritické faktory se rozprostřou,
- křivka očekávání má podstatně mělčí průběh,
- špatná cesta se odhalí podstatně dříve,
- chyby jsou levnější a rychleji se odstraní,
- možnost neúspěšný projekt zastavit včas.

Nevýhody:

- nezbytná týmová práce,
- neochota některých zákazníků se na nový způsob práce adaptovat,

- větší časová náročnost v průběhu vývoje na straně zákazníka.

2.6 Extrémní programování

Extrémní programování je označována za filozofii vývoje, vycházející z agilního přístupu. Tento přístup se i v současnosti vyvíjí na základě zkušeností.

Extrémní programování uznává základní čtyři hodnoty:

- udržování kvalitní komunikace mezi všemi zúčastněnými subjekty,
- jednoduchost při plnění aktuálních požadavků,
- funkční zpětná vazba na několika místech,
- odvaha programátora pustit se řešení významného problému, jehož řešení je jak časově náročné, tak způsobí další související problémy.

Extrémní programování obsahuje 12 základních praktik a tyto praktiky se dělí na tři základní oblasti. Business praktiky, týmové praktiky a programovací praktiky.

Business praktiky

Plánování hry, Zákazník na pracovišti, Vydávání malých verzí, Metafora

Plánování hry spočívá v tom, že si zákazník a vývojový tým určí, jak dosáhnout maximálního ekonomického přínosu daného softwaru za co nejkratší čas. Zákazník vytvoří seznam funkčním požadavků, které od systému očekávají. Každý požadavek se rozepíše na tzv. uživatelský příběh, který pak projde vývojový tým, a určí časovou a finanční náročnost. To se dále konzultuje se zákazníkem a požadavky jsou zákazníkem seskupeny do jednotlivých iterací dle důležitosti.

Zákazník na pracovišti představuje skutečného zákazníka, který je potřebný k vytvoření fungujícího softwaru. Tato osoba musí mít pravomoc určovat požadavky a stanovovat priority.

Vydávání malých verzí zrychluje zpětnou vazbu mezi programátory a zákazníkem. V extrémním programování je nutné vydávat nové verze co nejčastěji, aby se nové funkce dostaly k zákazníkovi co nejdříve a ten mohl provést zpětnou vazbu.

Metafora poskytuje zmapovaný systém jmen a popisů systému, které jsou jednoduché k pochopení celému cílu a díky kterým dokážou jednoduše komunikovat o systému.

Týmové praktiky

Párové programování, Společné vlastnictví kódu, Standardy kódu, Udržitelné tempo

Párové programování představuje, že na produkčním kódu vždy pracují společně dva programátoři u jednoho počítače, s jednou klávesnicí a myší. Programátoři si své role v páru

střídají a také se obměňují i páry. Pár většinou tvoří dva programátoři s různými vědomostmi, kteří se navzájem doplňují a kontrolují.

Společné vlastnictví kódu představuje skutečnost, že žádný z programátorů nemá výhradní právo na určitou část kódu či softwarový modul.

Standarty kódu jsou důležité pro dobrou spolupráci na společném kódu.

Udržitelné tempo musí být nastaveno tak, aby bylo udržitelné po dlouhou dobu. Tzn. okolo 40 hodin týdne

Programovací praktiky

Neustálá integrace, Jednoduchý návrh, Refaktorizace kódu, Testování

Neustálá integrace znamená, že do produkčního softwaru jsou integrovány nové funkce co možná nejčastěji, min. jednou denně.

Jednoduchý návrh spočívá ve vytvoření co nejjednoduššího programu, který splňuje aktuální požadavky, protože je pravděpodobné, že se požadavky na software stejně změní.

Refaktorizace kódu znamená neustálou kontrolu programátorů, zda by nešel kód zjednodušit či vylepšit. Tím se zvyšuje kvalita kódu.

Testování je vyznačováno tím, že programátoři extrémního programování začínají psaním testů a až následně vlastního funkčního kódu, který naplňuje požadavky dané testy. Zákazníci tvoří tesky akceptační. Testy v extrémním programování musí procházet na 100%.

2.6.1 Životní cyklus

Životní cyklus se projekt od projektu liší. Je to dáno tím, že se metodika přizpůsobuje podmínkám projektu a potřebám týmu.

Průzkum

Životní cyklus každého projektu začíná průzkumem. Při této fázi navrhují členové týmu jednotlivé architektury plánovaného systému, ze kterého pak vyberou tu nejvhodnější. V této fázi odhadují časovou náročnost projektu. Zákazník se v této části učí psát své požadavky v podobě karet zadání a v této etapě by měl sepsat požadavky na první verzi systému. Na základě požadavků nastává ověření znalostí a zkušeností programátorů, aby odpovídali náročnosti zadaného projektu. Pokud se znalosti jeví jako nedostatečné, musí si programátoři své nedostatky doplnit. Délka této etapy je závislá na znalostech a zkušenostech programátorů, technologiemi a oborem podnikání zákazníka. Tato fáze by neměla přesáhnout půl roku.

Plánování

Tato etapa navazuje na etapu průzkumu. Délka této etapy vychází z připravenosti zákazníka v první etapě. Pokud je zákazník dostatečně připraven, vyberou se jednotlivé karty zadání pro první verzi a dohodne se termín jejího dokončení. Tato etapa trvá mezi dvěma až šesti měsíci.

Iterace do uvolnění první verze

Tyto iterace trvají jeden až čtyři týdny. Požadovaným výsledkem je, aby z první iterace byla vytvořena kostra systému. O zadání dalších iterací rozhoduje zákazník na základě svých priorit. Výsledkem poslední iterace je první verze software určená k nasazení do provozu.

Zprovozňování

V této etapě dochází k nasazení a zprovoznění první verze systému do ostrého provozu. To následuje až po důkladném otestování a vyladění výkonu. Má k dispozici provozní hardware, což je velká výhoda. Iterace se při této etapě zkrátí ze tří na jeden den a současně dochází ke zpomalení vývoje. Každá změna musí být dobře promyšlena, protože je systém nasazen v ostrém provozu.

Údržba

Po zavedení systému do ostrého provozu dochází k viditelnému zpomalení vývoje. Jedním z důvodů je ten, že programátoři zajišťují vedle implementace také servis během aplikace.

Smrt

Smrt znamená konec vývoje systému, který může nastat ze dvou důvodů. Prvním, méně častým důvodem je ten, že je zákazník se systémem tak spokojený, že už nemá další požadavky na vyladění systému. Častějším důvodem smrti je ale skutečnost, že je vývojový tým neschopný přidat efektivně novou funkcionalitu. Při této závěrečné fázi se sepisuje rozsáhlý dokument o celém systému, který by měl týmu pomoci, když se bude po čase k systému vracet.

2.6.2 Výhody a nevýhody

Výhody:

- flexibilita při častých změnách požadavků až v samotném vývoji,

Nevýhody:

- riziko slepé uličky v důsledku toho, že se navrhuje pro dnešek a nemyslí se na zítřek,
- mezi plánováním se dostatečně neuvažují závislosti mezi jednotlivými úkoly,
- zdrojový kód a jeho test píše stejný člověk.

2.7 Dynamic System Development Method

Tato metoda není určena pouze pro vývoj software, ale pro vývoj systémů obecně. Proto všechny v ní uvedené procesy a kroky nejsou konkrétně popsány a je ponechána volnost při implementaci. Jedná se o agilní metodiku a úzce spojuje zákazníka a vývojový tým.

Metodika obsahuje 3 fáze vývoje a to pre-projektovou fázi, projektovou fázi a post-projektovou fázi. Druhá, nejrozsáhlejší fáze se dále člení na pět fází: analýza proveditelnosti, analýza business prostředí, vytvoření funkčního modelu, vytvoření designu a buildů a implementace.

2.7.1 Principy DSDM

Metodika DSDM je založena na devíti principech

Princip 1 Aktivní účast zákazníka

Je důležité zapojení všech uživatelů během vývoje, aby mohly být požadavky v průběhu vývoje detailně upřesnit.

Princip 2 Týmová pravomoc k rozhodování

Týmu musí být umožněno a tým musí umět rozhodovat o dílčích úlohách v průběhu vývoje. Je nutné stanovení přesných kompetencí, o čem smí tým rozhodnout a které rozhodnutí musí vykonat management.

Princip 3 Důraz na časté dodávky

Postup projektu není měřen množstvím času stráveného na úlohách, ale produkcí jednotlivých subdodávek.

Princip 4 Vhodnost systému pro podnikání

Zákazník očekává od systému určitý užitek, který bude spojen s jeho užíváním.

Princip 5 Iterační vývoj a inkrementální dodávky

Detailní specifikace projektu na začátku projektu je nemožná, protože obě strany nemají dostatečné znalostí. Řešením toho je inkrementální vývoj. Každý inkrement se konzultuje s konečným uživatelem, který jej musí odsouhlasit.

Princip 6 Vratné změny

Veškeré prováděné změny musí jít lehce vrátit do původního stavu.

Princip 7 Požadavky na vysoké úrovni

Na počátku projektu jsou požadavky na sytém určeny obecně, určují jenom cíl projektu a hranice. Detailní podoba je upřesněná až v dalším vývoji.

Princip 8 Průběžné testování integrované během celého vývoje software

Princip 9 Spolupráce

Jenom dokonalá komunikace a spolupráce může přinést úspěšně fungující systém.

2.7.2 Životní cyklus

Životní cyklus má sedm fází, přičemž první tři jsou sekvenční a další čtyři inkrementální a iterativní. Jednotlivé iterace mají předem stanovenou délku.

Úvod

Krátká etapa, sloužící k ujištění, že je projekt správně nakonfigurován a všechny procesy jsou správně nastaveny.

Studie proveditelnosti

V této fázi se zjišťuje, zda bude díky technickým možnostem projekt realizovatelný pomocí DSDM. Během této etapy vznikají dva dokumenty a to Zpráva o proveditelnosti a Rámcový plán vývoje. Délka této etapy musí být maximálně několik týdnů.

Obchodní studie

Zkoumají se zde základní charakteristiky fungování firmy zákazníka a požadovaných technologií. Při této etapě vzniká dokument Definice prostředí firmy, ve kterém jsou zapsány prováděné procesy a zainteresovaní lidé ve zkoumané firmě. Další dokumenty, které při této fázi vnikají je Definice architektury a Rámcový plán pro prototyp, kde je uveden prvotní návrh architektury vyvíjeného systému a navrhnutá strategie pro vývoj prototypu.

Stanovení modelu funkcí

Jedná se o první inkrementální a iterační etapu. Během iterací probíhá analýza i kódování. Jedním z dokumentů je Funkční model, tedy kód prototypu. Dalšími dokumenty jsou Priority funkcionalit, Revize funkčního modelu, Nefunkční požadavky a Analýza rizik dalšího vývoje. Během této etapy se prováděno průběžné testování.

Návrh a implementace

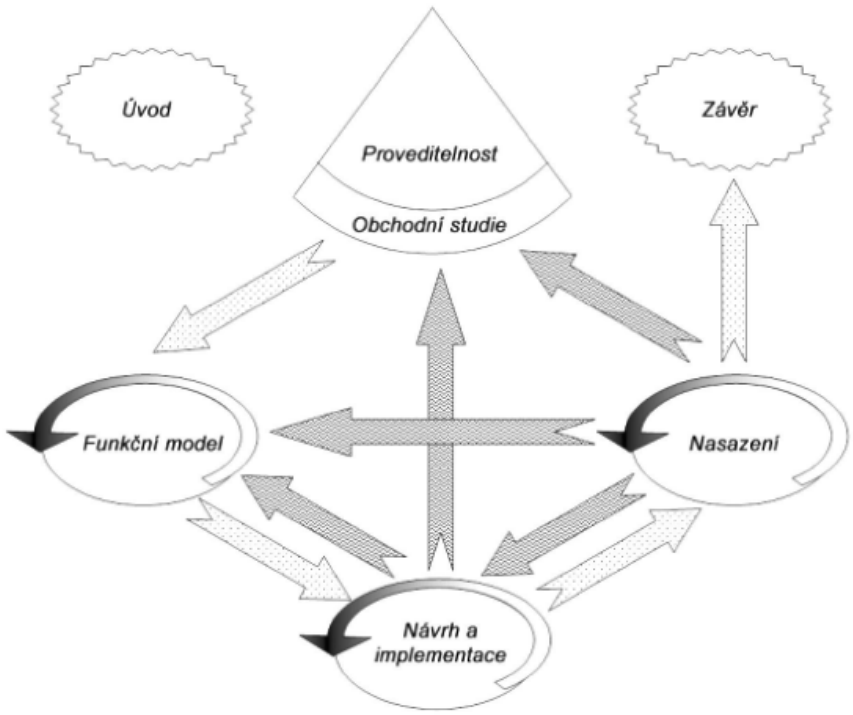
Navrhuje se a implementuje se vyvíjený systém. Dochází zde ke zpětné vazbě od zákazníka, vylepšuje se vzhled a celý systém.

Nasazení

V této etapě se systém nasadí do ostrého provozu u zákazníka. Probíhá zaškolení uživatelů na nový systém. Výsledkem této etapy jsou dokumenty Uživatelský manuál a Zhodnocení projektu.

Závěr projektu

Etapa závěr projektu následuje po nasazení nového systému u zákazníka a zjišťuje se jeho efektivita, a zda přináší zákazníkovi očekávaný užitek.



Obrázek 5 – životní cyklus DSDM [1]

2.7.3 Výhody a nevýhody

Výhody:

- dobře propracovaná metodika,
- velmi dobře dokumentovaná metodika,
- časové bloky a navržené úrovně priorit jednotlivých požadavků zaručí, že verze systému budou dodány včas,
- výhodou je použití metodiky DSDM spolu s metodikou RUP, XP.

Nevýhody:

- licencovaný přístup k této metodice omezuje výrazné rozšíření.

2.8 Crystal Method

Crystal Method není název jedné metodiky, ale představuje označení celé skupiny metodik, které vymyslel Alistair Cockburn. Jednotlivé metodiky jsou pojmenovány podle barev a navzájem se odlišují svou vhodností pro různě velké týmy a různě složité projekty. Čím je barva tmavší, tím je metodika robustnější a vhodnější pro rozsáhlejší, kritičtější projekty. Barvy metodik jsou čirá, žlutá, oranžová červená, hnědá, modrá a fialová.

Vhodná metodika se pro projekt volí na základě tří následujících vlastností projektu. První vlastnost je velikost vývojového týmu, druhá je kritičnost projektu a třetí je priorita projektu. Druhá vlastnost projektu může nabývat čtyř hodnot a to:

- ztráta komfortu (C) představuje nejmenší dopad, kdy lze systém nadále používat, ale uživatele to stojí větší úsilí,
- malá ztráta peněz (D) představuje větší dopad a zákazník ztrácí část financí,
- velká ztráta peněz (E) má větší dopad na ztrátu financí zákazníka,
- ztráta lidských životů (L) představuje nejhorší možný dopad.

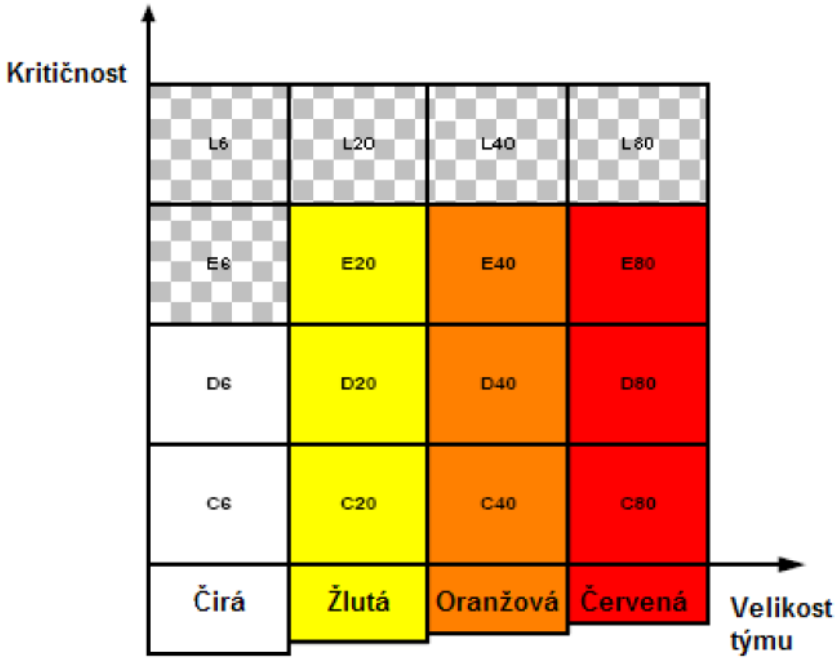
2.8.1 Principy metodiky Crystal Clear (CC)

Metodika CM vychází se sedmi principů, z nichž první tři jsou nezbytné pro fungování metodiky a další čtyři zvyšují úspěšnost celého projektu.

1. Pravidelné dodávky nových verzí, díky kterým se zvýší pravidelnost a frekvence zpětné vazby od uživatelů.
2. Zdokonalování procesů probíhá tím způsobem, že se občas sejdou všichni účastníci týmu a konzultují proces, jeho silné a slabé stránky a snaží se nalézt způsob, jak proces v dalších iterací vylepšit.
3. Podprahová komunikace je důležitý princip, objevující se i v jiných metodikách, který lze snadno realizovat jen v malých týmech a spočívá ve společné práci všech vývojářů na jednom místě. Každý člen týmu je tak přítomen a podprahově vnímá dění v týmu.
4. Přátelské prostředí, které souvisí s bezproblémovou komunikací všech členů týmu a které umožní prezentovat i nesouhlasné názory a upozornění na chyby svého kolegy.
5. Pozornost vychází z toho, že vývojář nemusí přeskakovat mezi úkoly pro různé projekty a minimálně dva pracovní dny práci pro jeden projekt, a pak může další dva dny dělat pro projekt druhý. Každý den má dvě hodiny, kdy nebude od své práce vyrušován.
6. Dostupnost uživatelů přestavuje potřebu, aby byl uživatel dostupný v dostatečně krátkém čase, kdy programátoři během vývoje potřebují ujasnit zadání.
7. Technické vybavení je pro tým samozřejmostí. Jenom díky kvalitnímu technickému vybavení je možné zajistit rychlý a kvalitní vývoj systému.

2.8.2 Životní cyklus

Životní cyklus CC je tvořen skupinou do sebe vnořených cyklů různých délek. Těmito cykly jsou vývoj, iterace, dodávka a celý projekt. Jejich vzájemné uspořádání je zobrazeno na obrázku níže.



Obrázek 6 – životní cyklus Crystal Method [7]

Projekt začíná etapou **mapování**. Během této etapy je sestaven tým. Tento tým je doplněn podle potřeby na dva až pět vývojářů. Následuje technika nastavení metodiky a na závěr se tvoří prvotní plán projektu.

Po etapě mapování následují časové cykly nazvané **dodávka**, která se skládá z jedné nebo více iterací. Na konci etapy je nový inkrement dodán alespoň některým uživatelům, kteří novou funkcionalitu otestují a informují tým o použitelnosti nových obrazovek. Během dodávky je alespoň jednou uskutečněn zdokonalující workshop, kdy je přezkoumán a vylepšen proces vývoje. Délka jedné iterace se může pohybovat mezi jedním týdnem až dvěma měsíci. A délka iteračního cyklu je závislá na schopnostech týmu. Čím delší je interval mezi jednotlivými iteracemi, tím je riziko problému větší. Posledním cyklem je **vývoj**, který je součástí integračního nebo denního cyklu.

2.8.3 Výhody a nevýhody

Výhody:

- disponují obrovskou flexibilitou,
- metodiky jsou orientovány více na lidi než na implementaci.

Nevýhody:

- dostupné jsou zatím jenom metodiky pro malé a středně velké týmy a pro lehké a středně obytné projekty,

- pro některé týmy může flexibilita představovat překážku, která spočívá v absenci striktně definovaných postupů.

3 Roztřídění metodik do skupin dle charakteristických společných znaků

Pojem metodika jako pracovní návod, který nám předepisuje, jak postupovat při určitém výkonu je již popsáno v předešlé části.

Metodik je nepřeberné množství a není daná jejich jednotná forma ani popis, tudíž je obtížné metodiky nějakým způsobem srovnat ani určit tu lepší metodiku pro naše potřeby. V zásadě lze metodiky rozdělit podle:

1. Zaměření metodiky – dělíme metodiky na globální a projektové.
2. Váhy metodiky – dělíme na těžké a lehké metodiky.
3. Rozsahu metodiky – dílčí metodiky a metodiky pokrývající celý životní cyklus vývoje.
4. Typu řešení - třídíme na vývoj nového řešení, rozšíření stávajícího řešení a integraci řešení.
5. Přístupu k řešení – dělíme na strukturované, objektové a rapidní vývoj.
6. Způsobu vývoje – na tradiční metodiky s vodopádovým cyklem a iterativní metodiky.

Výběr správného druhu metodiky a následně výběr konkrétní metodiky je složité. Musí se zvážit mnoho proměnných, které do výběru vstupují. Máme na mysli proměnné jako velikost projektů, organizaci a její kulturu, vývojové týmy, samostatné jednotlivce, byznys prostředí, rizika projektů atd.

K řešení jednotlivých softwarových projektů ve firmě se metodiky dělí dle zmíněné váhy a způsobu vývoje na tradiční (rigorózní) metodiky a metodiky agilní. Tradiční metodiky jsou oproti agilním metodikám formálně a striktně popsány. Agilní metodiky jsou vždy iterativní, kdy cykly neboli iterace, mohou být dlouhé jen několik dní, a jsou považovány za „lehké“.

U vývoje jakéhokoliv produktu je nejdůležitějším cílem docílení profitu z vyvíjeného produktu pro obě strany. Existuje mnoho přístupů, díky kterým lze k vývoji přistupovat, díky kterým může dosáhnout k různým výsledkům a různým naplněním cíle projektu. Mezi stěžejní rozdělení přístupů je rozdělení metodiky na klasické (rigorózní) a agilní, kterým se budu více věnovat v následujícím textu. V následujícím textu se budu podrobně věnovat rozdělení metodik podle způsobu vývoje. V současné době je hlavním cílem metodiky vytvořit rychlý a levnější software s využitím úzké spolupráce s budoucím uživatelem tak, aby byl software co nejefektivnější a splňoval všechny požadavky zákazníka. Tím se v nedávné době začaly zabývat metodiky nazývané jako agilní, které jsou v dnešní době na výslunní a je jim předpovídán obrovský rozvoj. V této části se proto budu zabývat bližším popisem právě

rigorózních a agilních metodik. V následující kapitole se budu blíže věnovat právě agilním metodikám a jejich srovnání.

3.1 Rigorózní metodiky

Mezi rigorózní metodiky se řadí:

- Vodopádový model
- Spirálový model
- Prototypování
- Rational Unified Process (RUP)

Rigorózní metodiky nazývané také tradiční metodiky se v dnešní době už moc nevyužívají, protože v tradičním pohledu na životní cyklus projektů se vyznačují velkou složitostí, striktní dokumentací, malé flexibilitě a nemožnosti reagovat na změny zákazníka během vývoje. Díky tomu dochází k častějším krachům projektů než je tomu u metodik agilních.

Rigorózní metodiky jsou poměrně striktní v definici procesů, činností, kompetencí, vytvářeného produktu a vycházejí z přesvědčení, že procesy při vývoji software lze detailně popsat, řídit a dále měřit. Proto bývají tak obsáhle popsány.

Nejdůležitější vlastností rigorózních metodik je fakt, že všechny specifikované požadavky zákazníka jsou řešeny postupně v určeném pořadí, dle předem sestaveného plánu a požadavky není možné během vývoje software měnit, jinak se musí projekt znovu přeplánovat.

Z nepřeberného množství rigorózních metodik jsem si vybral čtyři metodiky, které jsem detailně popsal v první kapitole. Jedná se o metodiku Vodopádový model, která je hlavním znakem rigorózních metodik, dále Spirálový model, Prototypování a metodiku RUP čili Rational Unified Process.

3.1.1 Hodnocení rigorózních metodik

Rigorózní jako klasické metodiky jsou využívány už řadu let. Nasazení vodopádového modelu životního cyklu do projektů bylo dávno před tím, než se vývoj SW projektů stal běžnou záležitostí. I když má tento model v dnešní době mnoho nevýhod, přináší se sebou i nezanedbatelné výhody. Vodopád je a zřejmě stále bude využíván jako metodika pro implementaci SW. Vodopádový model je využíván v těch projektech, kde se může maximálně využít jeho silných stránek a je možné bez problémů uplatnit tři základní principy jeho vývoje.

Tento model má i své nevýhody, které vyplývají z jeho použití. Problémy vznikají z aplikace tohoto životního cyklu na projekty, které nesplňují podmínky, že všechny požadavky na projekt a kompletní specifikace jeho cílů jsou známy už na začátku a významně se v době

vývoje projektu nemění. Tradiční vodopádový model se v současnosti používá velice málo nebo v nějaké modifikaci, která reflektuje soudobé požadavky na vývoj softwaru.

Jak je vidět, rigorózní metodiky mají i své stinné stránky, ale protože jsou nazvány tradiční metodikou, mají ve vývoji SW své místo. Jde o klasickou a historicky významnou metodiku, ze které vycházejí i soudobé metodiky a při řešení menších projektů ji lze použít beze změny.

3.2 Agilní metodiky

Mezi agilní metodiky patří:

- SCRUM
- Extrémní programování
- Dynamic System Development Method
- Crystal Method

Se zvyšováním dostupnosti výpočetní techniky se zvyšovala také dostupnost softwarových řešení. Projektů v informačních technologiích je stále více a jsou dostupné širšímu spektru lidí. Z počátku se využívaly klasické metody vývoje softwaru, tzv. „těžké“. Těmto metodám byla vytýkána jejich těžkopádnost, byrokratický přístup, příliš pomalá reakční doba a neefektivní využití zdrojů, tedy vývojářů. U těchto modelů jak je již známo vycházejí z předem daných požadavků, s jejichž změnou se v průběhu vývoje nepočítá. Proto každá změna a chyba v analytické části je velmi nákladná a může stát za celkovým neúspěchem projektu. Tyto skutečnosti napomohly v 90. letech 20. století vzniku agilních metodik, mezi které patří např. SCRUM, Extrémní programování, Dynamic Systém Development Method, a Crystal Method. Jednotlivé tyto metodiky jsou podrobně popsány v první kapitole.

Agilní metodiky se v angličtině označují jako „agile methodologies“, což se dá přeložit jako „živé“ metodiky. Dříve se užívalo označení „lightweight metodologies“, který měl zdůraznit lehkost metodik. Samotné překlady přesně vystihují cíle agilních metodik. Pokud chce firma obstát, musí být schopná dodat software v co nejkratším čase a držet se přitom minimálních nákladů. Je třeba si ale uvědomit, že při vývoji existují dva různé časy. První je celkový čas, který zabere realizace softwarového řešení, tzn. od počátečního sepsání smlouvy až po akceptaci konečného řešení. Druhým a významnějším časem je doba potřebná k dodání první fungující části software, kterou může zákazník používat při své práci. Čím dříve zákazník uvidí systém v provozu, tím dříve získává vývojový tým zpětnou vazbu. Z pohledu agilních metodik jsou totiž významnější lidé než dodržované postupy, produkované dokumenty nebo používané nástroje.

Agilní metodiky společně s tradičními procesovými metodikami, i přes odlišný přístup k řešení projektů, sdílí jeden hlavní cíl, čímž je skutečnost, aby byly vyprodukované projekty co nejúspěšnější a nejefektivnější.

Agilních metodik existuje mnoho a za jejich zrod se považuje tzv. Agilní Manifest, který na začátku roku 2001 sepsalo 17 softwarových inženýrů. Manifest obsahuje čtyři základní pravidla, která dodržují všechny agilní metodiky.

Základní pravidla Agile Manifesto:

1. Individualita a komunikace mají přednost před procesy a nástroji.
2. Fungující software je důležitější než rozsáhlá dokumentace.
3. Spolupráce se zákazníkem má přednost před vyjednáváním nad smlouvami.
4. Reakce na změnu je důležitější než striktní plnění plánů.

3.2.1 Agilní principy

Součástí agilního manifestu je 12 agilních principů:

- největší priorita je uspokojit zákazníka,
- vítané jsou změny a připomínky zákazníka v požadavcích i v pozdních fázích,
- preferování kratšího časového období k dodání funkčního software,
- důraz na každodenní spolupráci zákazníka s vývojáři,
- podpora jednotlivců při budování projektu,
- důraz na osobní rozhovor uvnitř týmu, jako nejúčinnější a nejefektivnější metodu zprostředkování informací,
- hlavním měřítkem pokroku je fungující software,
- podpora trvale udržitelného rozvoje,
- je kladen důraz na kvalitní technické vybavení,
- zjednodušení práce na práci, kterou je nutné udělat,
- samostatná organizace týmu.

V pravidelných intervalech tým analyzuje, jak být efektivnější a tomu uzpůsobuje své chování.

Jak je vidět, je tento postup stále zaměřen na týmy, ale je kladen důraz na jednotlivé členy týmu.

Agilní techniky rozbíjí projekt na malé části a postupně, inkrementálně budují požadovaný systém. Budování systému je nazýváno „inkrementální vývoj“ a jednotlivé části jsou označovány jako „iterace“, kde je každá iterace samostatným vývojovým procesem, který prochází všemi fázemi vývoje, od analýzy přes návrh až po testování.

3.2.2 Hodnocení agilních metodik

Agilní metodiky nelze chápat jako univerzální návod jak vytvořit software, ale musí se brát jako doporučení, rady a návody možných přístupů, který poskytují na základě svých dlouhodobých zkušeností autoři metodik.

Jednou z kladných vlastností bych hodnotil iterativní a inkrementální vývoj, který je výhodou. Jsou pevně určeny termíny dodání jednotlivých fází. V implementační fázi jsou metodiky rychlé a tak mohou vyvinout první verzi software ve velmi krátké době, který už lze komerčně využít. Další pozitivními vlastnostmi je kladení důrazu na efektivní komunikaci uvnitř týmu a na testování. Rizikem testování však může být skutečnost, že si programátor testuje svůj výtvor. Regresivní testování ale přináší týmu jistoty, protože se můžou najít chyby ve všech částech systému. S rostoucím systémem pak stoupá rozsah potřebných testů a to má za následek vzrůst nákladů na jeho tvorbu a údržbu. Agilní metodiky mají schopnost flexibilní reakce na změnu požadavků. Podmínkou jsou však zkušení a spolehlivý členové týmu. Toto je slabé místo metodik, protože na trhu práce nemusí být dostatečný počet uchazečů, disponující požadovanými zkušenostmi a znalostmi.

Agilní metodiky mají však i své záporné vlastnosti, kterými jsou absence formálnější smlouvy a omezující dokumentace. Pro agilní metodiku je nejdůležitější důvěra zákazníka v daný tým a ochota přistoupit na neformálně sepsanou smlouvu. Slabým místem agilních metodik je také fakt, že agilní metodiky je složitější realizovat v týmech, kde jsou zaměstnáni studenti na částečný pracovní úvazek, protože nejsou k dispozici po celou dobu a přicházejí tak o část komunikace.

Já osobně považuji za nejvýznamnější faktor ovlivňující úspěch projektu znalosti a zkušenosti všech zainteresovaných lidí v týmu, kteří mají mnohem větší vliv než používaná metodika a nástroje.

3.3 Srovnání agilních metodik s rigorózními

Hlavním rozdílem je ten, že klasické metodiky považují flexibilitu za fixní. Z toho vyplývá, že je na počátku projektu podepsána určitá specifikace, která je v dalších fázích projektu neměnná. Implementace této specifikace je pak časově a finančně náročná.

Agilní metodiky oproti tradičnímu přístupu zaměňují fixní a volné proměnné. Na počátku projektu se nejprve odhadnou a naplánují zdroje a čas, kterých bude při vývoji potřeba

a funkcionalita je ponechána jako volná proměnná, jejíž hodnota je upravována v závislosti úspěšnosti projektu a potřebě.

Nositel informace je zdrojový kód

Je to dáno tím, že agilní metodika produkuje mnohem méně dokumentace a vychází z toho, že základním, jednoznačným a spolehlivým nositelem informace je zdrojový kód.

Tuto myšlenku však nesdílí tradiční přístupy, které na objemné dokumentaci staví celý vývoj software.

Iterativní a inkrementální vývoj

Podle agilních metodik se plán projektu skládá z velmi krátkých iterací, během kterých se vyvíjí inkrementy výsledného software. V průběhu vývoje jsou zákazníkovi dodány jednotlivé fáze projektu a tím se výrazně zlepšuje zpětná vazba a eliminuje se riziko, že bude zákazník v závěrečné fázi nespokojený. Iterativní a inkrementální vývoj používají ve svých postupech také klasické metody, jejich iterace jsou ale poněkud delší.

Přímá osobní komunikace

U agilních metodik je přímá osobní komunikace jedním ze základních faktorů, jak vytvořit fungující a efektivní výsledný software. Agilní metodiky jsou přesvědčeny, že za mnoho komplikací, které vznikají během vývoje může špatná komunikace a problémové vztahy mezi členy týmu. Proto se snaží tyto komplikace eliminovat utužováním vzájemných vztahů a zavedením osobní komunikace do základních činností vývoje. Osobní komunikace je označována za nerychlejší a nejefektivnější způsob předání informací.

Oproti tomu, se v tradičních metodách klade důraz na činnosti než na vztahy mezi lidmi. Pokud vznikne nějaký problém, je jeho řešení přenecháno vedoucím pracovníkům. U tradičních přístupů je upřednostňována komunikace podle určených formálních pravidel než osobní komunikace.

Blízká spolupráce s uživatelem

Agilní týmy při své práci často komunikují s budoucími uživateli a upřesňují si tak potřebné informace a požadavky na software. Tato komunikace prostupuje celým projektem.

Oproti tomu tradiční metodiky komunikují se zákazníkem hned na počátku při psaní specifikace, v průběhu během občasných schůzek a nakonec během akceptace dodaného software. Komunikace u tradičních metodik je poměrně menší a nerovnoměrně rozložená.

Průběžné testování

Agilní metody kladou důraz na pravidelné regresní testování v průběhu celého projektu na základě častých změn ve vyvíjeném software.

Při tradičním přístupu etapa testování navazuje na etapu implementace. Je vyžadováno, aby testování a implementace prováděly různé osoby a snížila se tak pravděpodobnost opakování chybného porozumění specifikaci.

4 Porovnání metodik

Tato část diplomové práce je věnována právě agilním metodikám a jejich zhodnocení. Agilní metodiky jsem si pro porovnání vybral proto, že následující dvě části jsou věnovány právě na aplikaci agilních metodik ve společnosti SIEMENS, a proto jsem opustil od tradičních metodik a přešel k metodikám iterativním. Cílem této kapitoly je srovnání jednotlivých agilních metodik pomocí ukazatelů principy, role a postupy. Charakteristika a popis jednotlivých metod jsou vypsány v první kapitole. Jedná se o metodiky:

- Extrémní programování
- Dynamic System Development Method
- Crystal Method
- SCRUM

Cílem této kapitoly je získat přehled o jednotlivých agilních metodikách a jejich zhodnocení.

4.1 Porovnání vybraných agilních metodik podle principů

V této kapitole jsou porovnány principy, na kterých jsou jednotlivé metodiky postavené.

4.1.1 Extrémní programování

U Extrémního programování je nejdůležitější těchto šest principů:

1. **Předpoklad jednoduchosti**

   Cílem je vytvořit systém co nejjednodušší, aby byl lehce testovatelný, přehledný a snadný pro údržbu.

2. **Rychlá zpětná vazba**

   Čím rychlejší zpětná vazba zákazníka na vyvíjený produkt, tím dříve bude systém dokončen a implementován u konečného uživatele.

3. **Využití změny**

   Tento princip souvisí s principem jednoduchosti. Je pak možné rychle reagovat na změny a změněný požadavek implementovat do provozu.

4. **Přírůstková změna**

   Návrh, plán a velikost týmu se musí měnit postupně a nejde provést velkou změnu najednou.

5. **Kvalitní práce**

Hlavním východiskem u tohoto principu jsou kvalitní znalosti a zkušenosti jednotlivých členů týmu. Jen tehdy je možné produkovat kvalitní a efektivní výstupy.

6. **Malá počáteční investice**

   Nízké rozpočty nutí programátory a zákazníky k redukci svých požadavků a přístupů.

4.1.2 Dynamic System Development Method

Metodika vychází z devíti principů:

1. **Aktivní účast zákazníka**

   Pro zajištění kvalitní a přesné zpětné vazby jsou základním stavebním kamenem kvalifikovaní zástupci budoucích uživatelů nebo samotný uživatel, který bude k dispozici po celou dobu vývoje.

2. **Důraz na časté dodávky**

   Je důležité vydávat nové verze co nejčastěji. Jen tak se zrychlí zpětná vazba a objasní určité nedostatky systému.

3. **Týmová pravomoc k rozhodování**

   Tým má pravomoc rozhodnout o dalším vývoji produktu. Tím se sníží časová prodleva mezi jednotlivými etapami.

4. **Vhodnost systému pro podnikání**

   Je důležité splnit očekávání zákazníka a to vyvinout systém, který mu přinese očekávaný užitek.

5. **Iterační vývoj a inkrementální dodávky**

   Je vhodnější systém dodávat postupně než na konci celý projekt najednou na konci vývoje. Eliminují se tím chyby a nedostatky systému.

6. **Vratné změny**

   Podstatným krokem u vývoje je možnost, vrátit se do původního stavu.

7. **Požadavky na vysoké úrovni**

   Nejdůležitější pro tento princip jsou konkrétní a přesné požadavky, které urychlí rozhodování.

8. **Průběžné testování**

Aby se eliminovalo riziko, provádí se během celého vývoje produktu průběžné testování.

9. **Spolupráce**

   Dobrá spolupráce a bezproblémové vztahy na pracovišti jsou základem úspěšnosti.

4.1.3 Crystal Method

Metodika Crystal Method vychází ze sedmi principů:

1. **Pravidelné dodávky**

   Pravidelné dodávání nových verzí jsou důležité pro vývoj systému. Vzniká tím včasná zpětná vazba od uživatelů, která eliminuje rizika a nedostatky.

2. **Zdokonalování procesů**

   Čas od času je nezbytné, aby se celý tým sešel a prokonzultoval proces vývoje, našel silné a slabé stránky a nalézt řešení.

3. **Podprahová komunikace**

   Spočívá v tom, že pracují vývojáři na jednom místě a můžou efektivně sdílet své vědomosti.

4. **Přátelské prostředí**

   Častá a kvalitní komunikace jednotlivých členů týmů s nadřízeným a mezi sebou je pro vývoj nového software nezbytná. Je důležité umět říct svůj názor i nesouhlas.

5. **Pozornost**

   Vývojář má na starosti dva úkoly. Tyto dva úkoly jsou pro něj největší prioritou. Vývojář na jednom úkolu pracuje minimálně dva dny a pak může jít na úkol druhý.

6. **Dostupnost uživatelů**

   Je podstatné, aby byl zákazník dostupný v dostatečně krátkém čase.

7. **Technické vybavení**

   Pro dosažení nejlepšího produktu je nejdůležitější kvalitní technické vybavení a osvědčené postupy.

4.1.4 SCRUM

Metodika SCRUM vychází z těchto principů:

1. **Neexistuje** žádný manažer v klasickém pojetí.

2. **Aktivity** týmu řídí SCRUM master.
3. **Tým** se podílí na plánování.
4. **Úkoly** se nezadávají, úkoly si každý vybere sám.
5. SCRUM **schůze** se svolává každý den a trvá 15 minut.

Porovnání metodik podle principů			
Extrémní programování	**DSDM**	**Crystal Method**	**SCRUM**
Rychlá zpětná vazba	Aktivní účast zákazníka	Dostupnost uživatelů	Zpětná vazba zákazníka
Přírůstková změna	Časté dodávky	Pravidelné dodávky	Výběr úkolů členem týmu
Kvalitní práce	Týmové rozhodování	Podprahová komunikace	Týmové plánování
Využití změny	Iterace a inkrementace dodávek	Zdokonalování procesů	
Předpoklad jednoduchosti	Průběžné testování	Technické vybavení	
Malá počáteční investice	Spolupráce	Přátelské prostředí	
	Vratné změny	Pozornost	

Tabulka 1 – srovnání podle principů

4.1.5 Srovnání

U všech čtyř srovnávaných metodik se principy nijak neliší, jelikož všechny vycházejí z agilního vývoje software. U metodiky Extrémní programování převažuje princip odpovědnosti, protože jim úkoly nejsou přidělovány, ale vybírají si je sami. U Crystal metodiky je jako u jediné zajištěn nerušený čas, což se od ostatních metodik odlišuje tak, že se nespoléhá na pomoc jiného programátora.

4.2 Porovnání agilních metodik podle rolí

Role, podle kterých jsou porovnány jednotlivé metodiky

4.2.1 Extrémní programování

U extrémního programování existují následující role:

- Kouč

  Má na starosti celý tým. Vysvětluje proces manažerům na vyšších úrovních a pomáhá programátorům je technickými dovednostmi jako testování a refaktorizace.

- Stopař

  Sleduje postavení celého týmu a hlídá projekt, zda se vyvíjí podle plánů a přání zákazníka. Roli kouče a stopaře má na starosti většinou jeden člověk.

- Programátor

  Programátor je základní stavební kámen týmu.

- Zákazník

  Jedná se o osobu, která píše karty zadání a upřesňuje nedostatky.

- Tester

  Role testera pomáhá zákazníkovi vybírat a psát testy funkcionality. Zajišťuje chod všech testovacích nástrojů a má za úkol pravidelné spouštění testů funkcionality a seznámení členů týmu s výsledky těchto testů.

- Konzultant

  Řeší minimálně s jedním programátorem specifické problémy, na které čas od času narazí.

- Velký šéf

  Zodpovídá za vývojový proces jako celek. Má na starosti zajištění dostatečného počtu členů v týmu.

4.2.2 Dynamic System Development Method

Role u DSDM jsou:

- **Vývojář**

  Mají na starosti samotný vývoj systému. Jedná se o zaměstnance, které mají na starost analýzu, návrh, programování a testování vyvíjeného systému.

- **Senior vývojář**

  Na seniora vývojáře může být povýšen některý z vývojářů, pokud disponuje s vyššími zkušenostmi než jeho kolegové.

- **Technický koordinátor**

  Technický koordinátor navrhuje architekturu vyvíjeného systému a je zodpovědný za její technickou kvalitu.

- **Zástupce uživatelů**

  Úkolem je zajišťovat zpětnou vazbu budoucího uživatele s programátory.

- **Rádce**

  Ve specifických požadavcích zastupuje zástupce uživatele.

- **Vizionář**

  Osoba, která přišla s novou myšlenkou nového systému.

- **Výkonný ředitel**

Osoba, která je na straně zákazníka a je zodpovědná za financování celého projektu.

- **Projektový manažer**

  Má na starosti bezproblémový průběh projektu.

- **Vedoucí týmu**

  Má na starosti celý tým a kontroluje jejich práci a komunikuje s členy.

- **Tester**

  Stará se o jednotlivé testy

- **Písař**

  Účastní se všech porad a schůzek a zapisuje všechny požadavky. Zodpovídá za dokumentaci.

- **Specialisté**

  Experti na různé oblasti vývoje software.

4.2.3 Crystal Method

Role Crystal Method jsou:

- **Sponzor**

  Má na starosti financování projektu.

- **Uživatel**

  Zástupce budoucího uživatele, který komunikuje s vývojovým týmem a testuje funkční vlastnosti.

- **Vývojář**

  Programují jednotlivé navrhování.

- **Hlavní vývojář**

  Jedná se o osobu, která je na vyšší úrovni od vývojáře. Disponuje vyššími kvalifikačními dovednostmi a znalostmi.

- **Koordinátor**

  Koordinuje práci ostatním.

- **Tester**

  Testuje vyvíjený systém.

- **Písař**

  Tvoří uživatelskou dokumentaci.

4.2.4 SCRUM

Scrum definuje tři role, které se podílejí na vývoji procesu:

- **Scrum Master**

  Osoba, která se podobá vedoucímu týmu. Odstraňuje překážky bránícímu vývoji a vytváří prostředníka mezi týmem a vnějším prostředím.

- **Product Owner**

  Jedná se o zákazníka. Určuje pořadí vlastností, které budou vyvíjeny během jednotlivých sprintů a stanovuje jejich priority. Má pravomoce tyto priority měnit z důvodu maximálního zisku.

- **Tým**

  Tým tvoří samotní vývojáři.

Porovnání metodik podle rolí			
Extrémní programování	**DSDM**	**Crystal Method**	**SCRUM**
Kouč	Vedoucí týmu	Koordinátor	Vedoucí týmu
Stopař	Vývojář	Hlavní vývojář	Zákazník
Programátor	Senior vývojář	Vývojář	Tým
Zákazník	Zástupce uživatelů	Uživatel	
Tester	Rádce	Sponzor	
Konzultant	Výkonný ředitel	Tester	
Velký šéf	Tester	Písař	
	Technický koordinátor		
	Vizionář		
	Projektový manažer		
	Písař		

Tabulka 2 – srovnání podle rolí

4.2.5 Srovnání

U všech metodik lze určit tři skupiny rolí: programátorské, uživatelské a řídící. U programátorské skupiny se rozlišují programátoři podle zkušeností. U Extrémního programování má kouč na starosti celý tým a své znalosti pomocí párového programování rychle předává mezi ostatní a zvyšuje tím technickou zastupitelnost. Hlavní vývojář u Crystal Clear je nezastupitelný. Své znalosti o architektuře nepředává a v případě problémů bývá hodně zatížen.

Je proto důležité, aby si zaučil alespoň jednoho náhradníka, který ho zastoupí. U metodiky SCRUM neexistuje žádný vedoucí, SCRUM se řídí sám a spoléhá na profesionalitu členů týmu.

Uživatelských rolí je nejvíce u DSDM, z toho plyne, že klade metodika důraz na vazby mezi týmem a zákazníkem. Jedním z mnoha důvodů je skutečnost, že DSDM se podílí i na větších projektech, kde se vývoje účastní více týmů. Role tester se vyskytuje ve všech metodikách. Je to dáno tím, že jedna ze společných vlastností agilních metodik je regresivní automatizované testování.

4.3 Porovnání agilních metodik podle postupů

V této části se budu zabývat postupy, podle kterých se řídí jednotlivé agilní metodiky.

4.3.1 Extrémní programování

Extrémní programování je postaveno na 12 principech. Tyto postupy tvoří dohromady jeden pevný celek a při použití jednotlivě by se mohly vyskytnout problémy, které by ohrozily daný projekt.

- **Plánovací hra**

  Hlavní roli zde hraje zákazník, který stanovuje priority, určuje další požadavky, rozhoduje o šíři celkového projektu a upravuje datum uvedení jednotlivých verzí do provozu.

- **Malá verze**

  Pro zákazníka je podstatné, dodávat malé verze v krátkých časových intervalech**.**

- **Metafora**

  Zajišťuje komunikaci mezi vývojovým týmem a zákazníkem.

- **Jednoduchý návrh**

  Software má mít neustále nejjednodušší formu, která pokrývá všechny požadavky zákazníka.

- **Testování**

  Pomocí testů se ověřují zdrojové kódy. Díky testům se programátoři dovědí, jak je nová verze vyvinutá a co jí ještě schází.

- **Refaktorizace**

  Programátoři udržují zdrojové kódy tak, aby neobsahovaly duplicity, aby zvýšili čitelnost zdrojového kódu a aby šlo lehce přidat novou funkčnost.

- **Párové programování**

U Extrémního programování se zdrojový kód píše v páru. Jeden programátor píše kód, test nebo redaktoruje a druhý jeho činnost kontroluje.

- **Společné vlastnictví**

  Všichni členové týmu nesou stejnou odpovědnost za celý vyvíjený systém.

- **Nepřetržitá integrace**

  Znamená, že se procesy testování a integrace opakují ve velmi krátkém intervalu. Alespoň jednou za den.

- **40 hodinový týden**

  Pracovní týden by neměl překročit 40 hodin a přesčasy jsou u vývoje software nevítané. Přesčasy jsou označovány ohnisko problému, a proto Extrémní programování zakazuje přesčasy v dvou následujících týdnech.

- **Zákazník na pracovišti**

  Potřeba mít zákazníka kdykoliv k dispozici.

- **Standardy pro psaní zdrojového kódu**

  Tyto standardy jsou nutné kvůli společnému vlastnictví kódu. Všichni členové týmu je musí znát a dodržovat.

4.3.2 Dynamic System Development Method

Postupy DSDM

- **MoSCoW**

  Je to složenina čtyř písmen označující úrovně priorit. Jde o postup, díky kterému DSDM přiřazuje k jednotlivým požadavkům různé priority, podle kterých je během vývoje s požadavky nakládáno. Priority se dělí na nezbytné, doporučené, možné a odložitelné.

- **Časový blok**

  Znamená, že celý projekt i jednotlivé iterace mají pevně stanovený termín ukončení, které nelze za žádných okolností odložit. Délka iteračních bloků je v rozmezí od dvou do šesti týdnů, dle náročnosti iterace. Během bloku se postupně realizují požadavky, seřazené podle priorit. Žádná iterace se nedá přeskočit a tak se může stát, že se nestihnou realizovat méně důležité požadavky. Každý blok projde postupně třemi fázemi: zkoumáním, zdokonalením a konsolidací. První fáze kontroluje, zda projekt postupuje k cíly. Druhá fáze se podílí na zdokonalování procesů a poslední k dokončení všech nedostatků.

4.3.3 Crystal Method

V metodikách Crystal Method se místo o praktikách hovoří jako o strategiích a technikách. Metodika ale ponechává volnost a nediktuje, jakou taktiku nebo strategii použít. Jde o pět strategií a sedm technik.

- **Průzkum**

  Na počátku se zkoumá, má-li projekt vůbec smysl. Zda bude ekonomicky efektivní a bude provozovaný za rozumných ekonomických podmínek. Zda tým disponuje dostačující technologií a zkušenostmi. Průzkum trvá od několika dnů až dva týdny a podle výsledku je projekt buď spuštěn, nebo zamítnut.

- **Motivující úspěch**

  Lidi v jejich práci motivuje již dosažený úspěch. Proto se začíná s realizací nejjednodušších požadavků, které zaručí úspěch.

- **Běžící kostra**

  Cílem je vytvořit co nejdříve fungující kostru budoucího systému. Jejím úkolem je propojit jednotlivé části systému, čímž se zvýší možnost paralelního vývoje v následujícím období.

- **Inkrementální architektura**

  Nutnost běžící kostru zdokonalit. To se realizuje pomocí inkrementálních změn architektury navrhovaného systému.

- **Informační tabule**

  Na informačních tabulích se prezentuje průběh celého projektu a jsou umístěny na snadno dostupných místech.

- **Nastavení metodiky**

  Na počátku projektu je nutné nastavit prvotní parametry metodiky.

- **Zdokonalující workshop**

  Tým se čas od času sejde, a probere své zkušenosti a názory na probíhající proces. Setkání probíhá vždy po uvolnění nové verze. Konzultují se následující postupy a navrhují postupy nové.

- **Bleskové plánování**

  Plánování se používá pro horizont následujících třech měsíců.

- **Věštění doby trvání**

  Pokud není jiná možnost, jak již na počátku určit dobu trvání jednotlivých úkonů, čas se pouze odhaduje. Odhaduje se také počet tříd, náročnost a potřebné znalosti vývojářů.

- **Denní schůzky**

  Pochází z metodiky SCRUM. Spočívá v tom, že se každý den ráno celý tým na pár minut sejde a konzultují daný projekt.

- **Optimalizace rozhraní**

  Je nutné identifikovat a určit jednu nebo dvě nejdůležitější skupinu uživatelů a pro uživatele těchto skupin optimalizovat rozhraní vyvíjeného systému.

- **Miniprojekt**

  Je určen pro nového člověka, který nastoupil do týmu a ještě není seznámen s celou závislostí jednotlivých procesů.

- **Blízké programování**

  Lehčí verze párového programování.

- **Přírůstkové grafy**

  Technika je určená k jednoduché, rychlé a přehledné prezentaci pokroku a vývoji. Graf je umístěn na viditelném místě a je pravidelně aktualizován.

4.3.4 SCRUM

Procesy SCRUMU jsou podporovány třemi artefakty:

- **Product Backlog**

  Jedná se o seznam všech požadavků na vyvíjený produkt, jeho vlastnosti a změny. Product Backlog je v průběhu vývoje průběžně doplňován a udržován.

- **Sprint Backlog**

  Z Product Backlog jsou ze začátku každého sprintu vybrány nejdůležitější požadavky a z těch je tvořen Spring Backlog. Jednotlivé požadavky jsou rozděleny na menší úkoly, snadněji splnitelné. Jednotlivé úkony si vybírají sami vývojáři.

- **Burn Down Chart**

Výchozím znakem je Brun down graf, který zobrazuje počet dosud nesplněných úkolů v čase. Osa X zobrazuje datum, zatím co osa Y zobrazuje nesplněné úkoly. Graf by měl mít klesající lineární průběh.

Porovnání metodik podle postupů			
Extrémní programování	**DSDM**	**Crystal Method**	**SCRUM**
Plánovací hra	MoSCoW	Průzkum	Product Backlog
Malá verze	Časový blok	Motivující úspěch	Sprint Backlog
Metafora		Běžící kostra	Burn Down Chart
Jednoduchý návrh		Inkrementální architektura	
Testování		Informační tabule	
Refaktorizace		Nastavení metodiky	
Párové programování		Zdokonalující workshop	
Společné vlastnictví		Bleskové plánování	
Nepřetržitá integrace		Denní schůzky	
40 hodinový týden		Miniprojekt	
Zákazník na pracovišti		Blízké programování	
Standardy zrojového kódu		Přírůstkové grafy	

Tabulka 3 – srovnání podle postupů

4.3.5 Srovnání

Postupy DSDM časový blok a MoSCoW jsou alternativní k plánovací hře v Extrémním programování a bleskovému plánování v Crystal Method. Všechny slouží k plánování rozsahu verze na základě odhadovaného času a přidělené priority. Odlišují se však ve způsobu. Nejsnadnější k pochopení a použití je plánovací hra, avšak bleskové plánování je přehlednější.

U metodiky Extrémní programování jsou neobvyklými postupy metafora a společné vlastnictví. Není typické, že je zdrojový kód vlastnictví všech a může ho kdokoliv a kdykoliv změnit. Podle mého názoru by bylo výhodnější, kdyby byl za kód někdo zodpovědný, ale oproti tomu je jisté, že pokud by se mělo Extrémní programování zrychlit, nemůže se čekat, až chybu opraví někdo jiný. Jediný kompromis je v tom, že kód může změnit kdokoliv, ale bude existovat nějaká pověřená osoba, která musí změnu kódu přijat a schválit. U metodiky Crystal Method bych se chtěl zmínit o postupu miniprojekt. Výhodou tohoto postupu je, že seznámí nové členy týmu s projektem. Částečně by se dal přirovnat k párovému programování se zkušeným partnerem. Velkou podobnost jsem nalezl také v párovém a blízkém programování.

5 Použití konkrétní metodiky na projektu ve firmě SIEMENS

5.1 Představení společnosti SIEMENS a popis projektu ESS portál

Středisko globálně sdílených služeb, tzv. Global Shared Services Accounting&Finance Center v Praze a pobočkou v Ostravě (GSS A&F Center) nabízí podnikům ze skupiny Siemens v regionu Česká Republika, Německo a severozápadní Evropa rozhodující přednosti prostřednictvím standardizovaných účetních služeb.

Seskupování účetních aktivit, výhodné náklady lokalit jakož i kontinuální stupňování produkce poskytují značný přínos k cíli firmy, což je „globální konkurenceschopnost".

GSS A&F centrum Ostrava obsluhuje následující portfolio:

- Převzetí pracovně náročných, repetičních, avšak kvalifikovaných účetních operací jako např. uzávěrky a služby pro dlužníky, věřitele a účetnictví hmotného investičního majetku atd.
- Převzetí účetních činností z oblastí pro region Česká Republika, Německo a severozápadní Evropa.
- Převzetí e-CISu, účetnictví a dalších témat z pracovních oblastí SASM (Globálně sdílené služby Mnichov).

GSS A&F Center je vedena pod správou Corporate Finance. Tímto je zaručena řádnost, neutralita a důvěrnost.

Popis projektu ESS portál

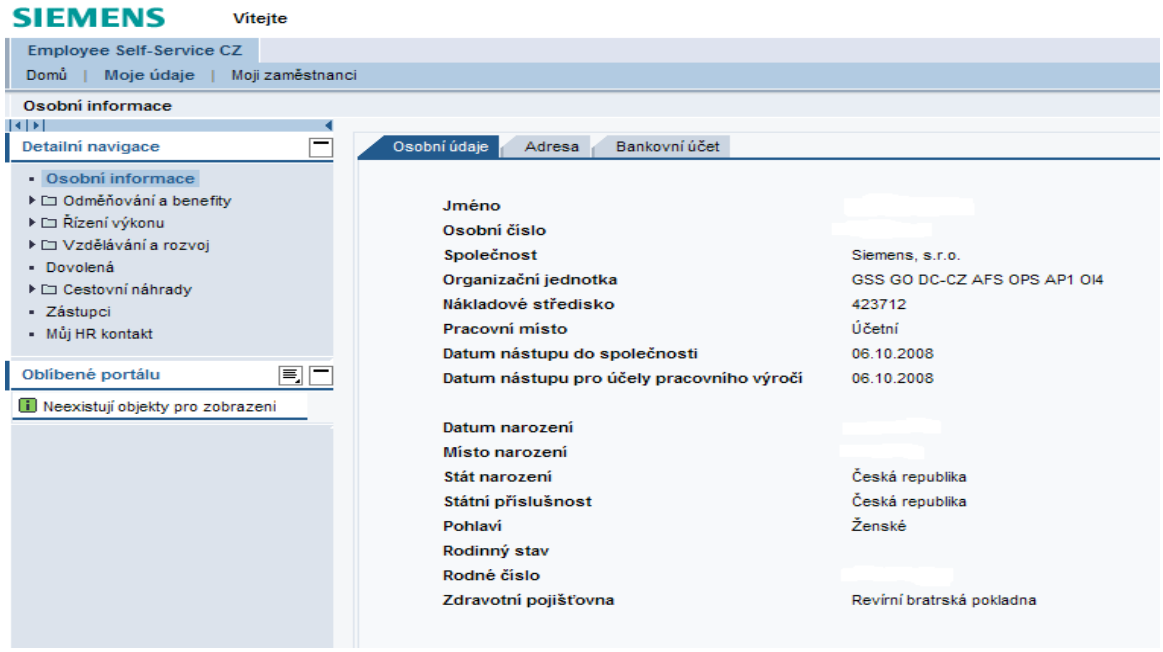
Cílem ESS portálu bylo zjednodušení administrativy a odstranění zbytečného „papírování". Hlavní výhodou je absence tištěných formulářů, to vše se vyplňuje přes příslušnou žádost v ESS portálu.

Portál má dvě části – zaměstnaneckou a manažerskou, které se trochu liší svými funkcemi. Druhá část je přístupná pouze manažerům, kteří v ní najdou dodatečné nástroje na schvalování a reporty.

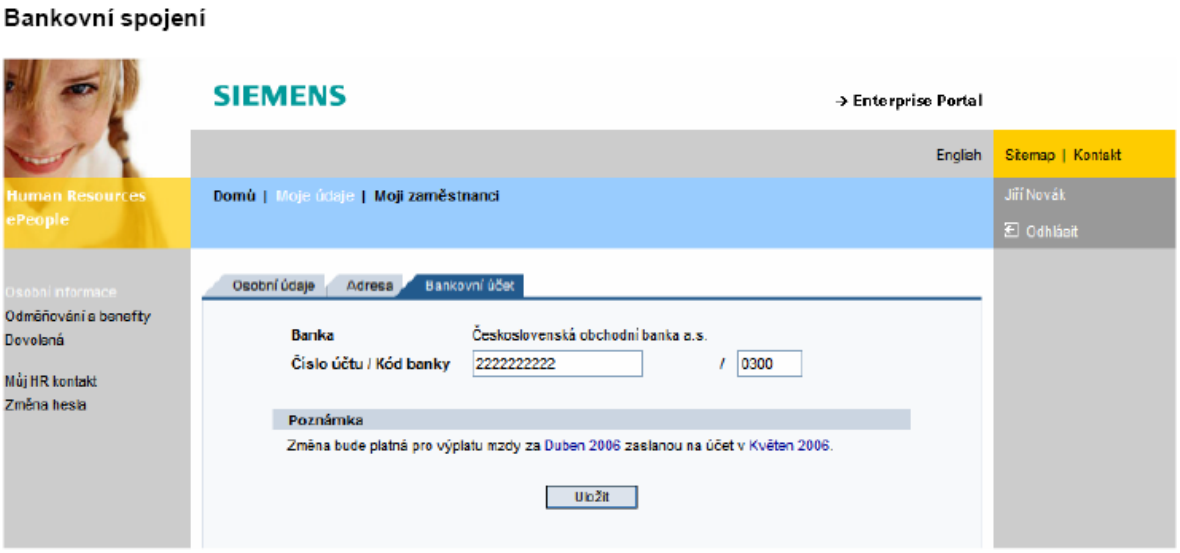
Funkce na ESS portálu:

1. **Moje údaje**

Osobní informace – díky této funkci se nebude muset každá změna jak bydliště, tak bankovního konta apod. nikam posílat, ale bude možné si ji změnit přímo na ESS portálu.



Obrázek 7 – ukázka ESS CZ

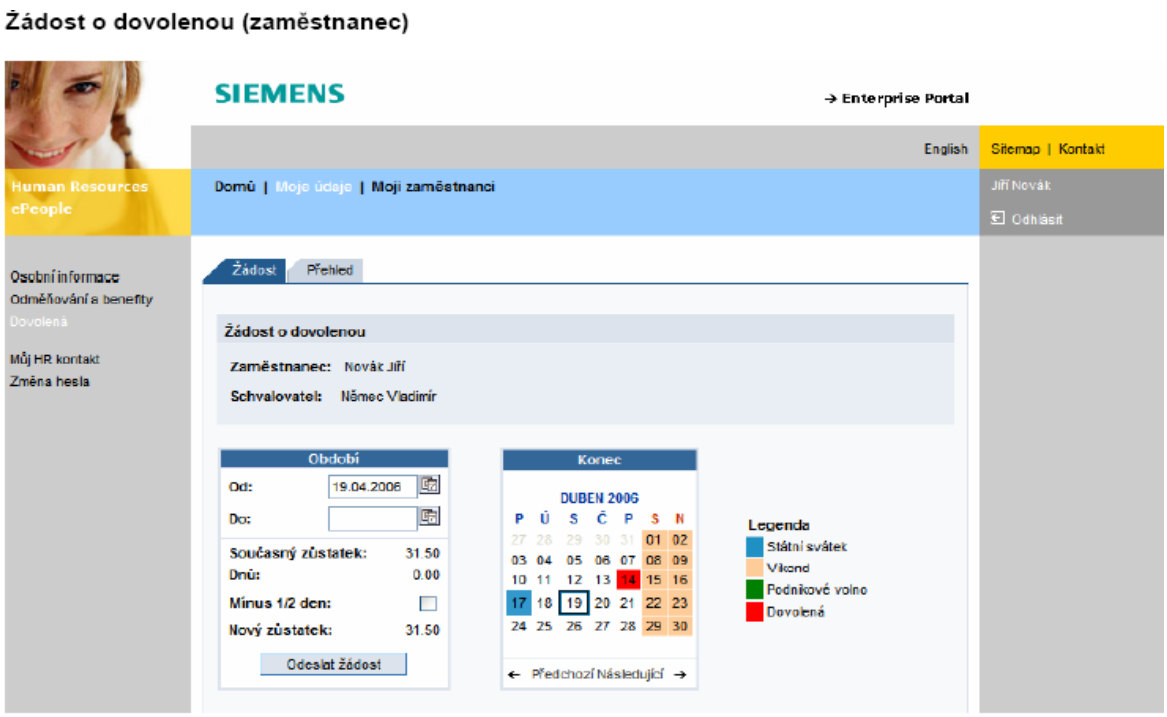


Obrázek 8 – ukázka ESS Cz Bankovního spojení

Výplatní páska – hned po uzávěrce bude výplatní páska v elektronické podobě zobrazena v sekci Odměňování a benefity pod položkou Výplatní páska.

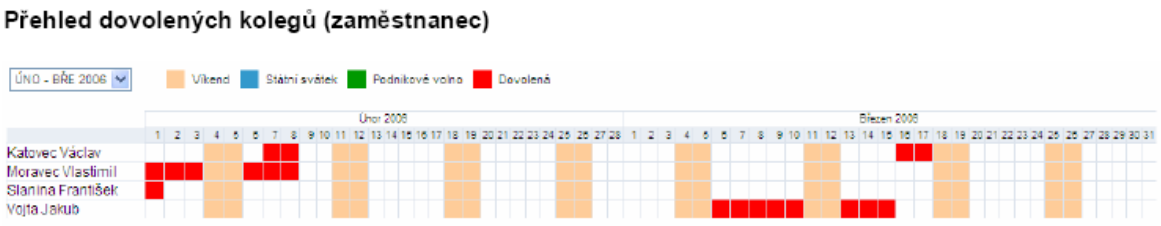
Benefity – je odkaz na externí aplikaci, kde si můžou zaměstnanci vybrat z celé řady benefitů neboli zaměstnaneckých výhod.

Dovolená – je možné si požádat o dovolenou přímo na ESS portálu v sekci Dovolená a odpadne tím tisk Dovolenky. Nadřízený je automaticky dostane ke schválení a je zabezpečeno, aby čerpání proběhlo v pořádku.



Obrázek 9 – ukázka ESS portálu, Dovolená

V této sekci je také k dispozici zobrazení přehledu dovolených ostatních kolegů.



Obrázek 10 – ukázka ESS portálu, Přehled dovolených

Potvrzení příjmů je další sekce na ESS portálu. Díky této aplikaci si zaměstnanec vybere, jaký druh potvrzení potřebuje a odešle přes ESS portál. Mzdová účetní pak zařídí vše potřebné a zašle potvrzení poštou.

Řízení výkonu – v tomto modulu najdou zaměstnanci aplikace na řízení pracovního výkonu čímž je Popis pracovního místa a Bonusové cíle. Zobrazen je pouze zaměstnancům, kterým byl zpřístupněn.

Můj HR kontakt – v této sekci najde zaměstnanec oprávněnou osobu, na kterou se může obrátit s nějaký problémem týkající se HR.

1. **Moji zaměstnanci (přístupné pouze pro vedoucí zaměstnance).**

Organizační struktura – tato část zobrazuje organizační strukturu svého oddělení.

Dovolená – zde uvidí nadřízení zaměstnanec přehled všech dovolených týmu a zároveň zde tyto dovolené schvaluje či zamítá.

Řízení výkonu – v této sekci schvaluje žádosti zaměstnanců v modulu řízení výkonu, což jsou Bonusové cíle a Popis pracovního místa.

Cestovní náhrady – tento modul umožňuje snadnou správu pracovních cest a drobných vydání. Jednoduchým workflow bude cesta povolena a následně ji po návratu bude možné vyúčtovat.

Zástupy – v této sekci si může nadřízený pracovník nastavit své zástupy na celou paletu činností a taky možnost nastavení délky trvání tohoto zástupu.

Můj HR kontakt dává informaci, na koho se může obrátit v záležitostech týkající se HR.

Vzdělání a rozvoj dává přehled o plánovaných a uskutečněných školeních na příslušný rok včetně přehledu povinných školení.

5.2 Role SCRUM při vývoji ESS portálu

Metodika SCRUM definuje tři role, které se na celém procesu vývoje podílí.

K dispozici je jeden **tým**. Členové týmu byli vybráni tak, aby nevznikaly problémy v komunikaci se zákazníkem a v týmu. Ne všichni členové týmu měli zkušenosti s agilní metodikou, konkrétně s metodikou SCRUM. Aby byly eliminovány problémy, které by mohli z důvodu nedostatků informací o metodice SCRUM během vývoje vzniknout, byli všichni členové týmu proškoleni na agilní principy a samotnou metodiku SCRUM. Tým se skládal ze dvou programátorů, dvou grafiků a dvou implementátorů grafického systému a jednoho testera. U všech členů týmu byla prokázána dobrá komunikativnost, což je důležité jak při telefonické komunikaci se zákazníkem, tak při komunikaci mezi samotnými členy týmu. Pracovníci týmu nesdíleli stejnou kancelář, takže komunikace uvnitř týmu byla omezena.

Za **Scrum Master** byla zvolena osoba, která disponuje dobrými organizačními schopnostmi, což je důležité při vytváření dobrého pracovního prostředí. Tato osoba měla sloužit jako prostředník mezi týmem a vnějším světem. Dohlížela na dodržování postupů Scrum a nad úkoly jednotlivých členů týmu. Zajišťovala odstranění překážek bránícímu vývoji a dohlížela nad tím, aby nebyli členové přetěžováni. Tato osoba byla poznačena v databázi zakázek a v případě žádosti o informaci o dané zakázce, dodávala potřebné informace.

Product Owner je označení zákazníka, což byl v tomto případě firma SIEMENS. Úkolem zákazníka bylo zadání cíle projektu a určit jednotlivé priority. Bylo potřeba, aby byl zákazník týmu stále nablízku, aby se zrychlila zpětná vazba. Určení priorit požadavků sdělil Product Owner na úvodní osobní schůzce, změny v prioritách a výtky se řešily pak pouze

telefonicky, v méně častých případech přes e-mail. Způsob získávání informací bylo položeno na vhodném dotazování, co je pro zákazníka více podstatné a co by chtěl nejdříve.

Celý proces vývoje se dělil na **„sprinty“**, což bylo období dlouhé 2-3 týdny. Na začátku každého sprintu se tým dohodl, na kterých vlastnostech budovaného softwaru bude pracovat a v průběhu iterace, se soustředil pouze na ně. Výsledkem každého sprintu byl tedy stabilní program, připravený k předání zákazníkovi.

5.3 Zahájení projektu a průběh při vývoji ESS portálu

Před zahájením projektu se rozhodovalo, zda má projekt pro firmu smysl a zda jej chce realizovat. Bylo třeba zvážit realizovatelnost, komplexnost, časovou náročnost a prioritu s ostatními projekty. Před začátkem každého sprintu se stanovil jeho průběh a očekávaný výsledek. K tomu sloužil **Sprint Planning Meeting**. Na počátku zakázky byla stanovena jeho pracnost a rozplánování do jednotlivých sprintů s přihlédnutím k výhledovému měsíčnímu plánu. Byl stanoven předběžný interní termín dokončení a termín odevzdání zákazníkovi, v němž byla zahrnuta i rezerva. Předběžný interní termín byl stanoven na 15 týdnů i s rezervou. Byl vytvořen **Product Backlog**, seznam všech požadavků na vyvíjený produkt. V Product Backlog byly jednotlivé požadavky ohodnoceny podle priority a seřazeny. Tím vznikl první odhad vývoje. Dále byl ohodnocen každý bod časovým odhadem náročnosti. Tento krok byl v počátečním stádiu projektu těžší. Tento seznam byl během jednotlivých sprintů postupně doplňován a měněn. Product Backlog byl v rukou Product Ownera, který určoval priority jednotlivých požadavků, a členové týmu pak doplnili časový odhad pro každou položku. Na Sprint Planning Meetingu byly sprinty seřazeny podle priorit a rozsahu. Jednotlivý členové týmu si následně vybrali Story dle jejich pořadí a story rozložily na samostatné úkoly. Sprint Planning Meeting vždy trval průměrně 3,5 hodiny. Z každé schůzky byl pořízen zápis.

Následně byly z Product Backlog vybrány požadavky s nejvyšší prioritou a byl z nich vytvořen **Sprint Backlog**. Požadavky byly rozděleny na menší úkony a tyto úkoly si členové týmu vybírali sami. Dá se říct, že Sprint Backlog měl na starosti pouze vývojový tým.

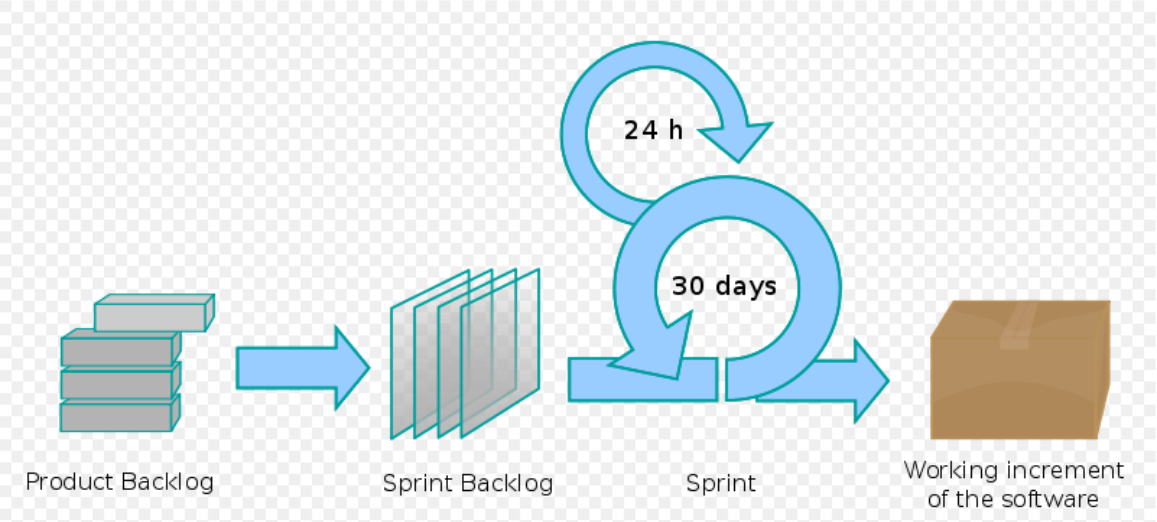
Na počátku každého dne se svolal **Daily Scrum**. Jedna se o každodenní setkání všech zainteresovaných členů včetně Scrum Mastera a Product Ownera, které probíhalo vždy v 9:00 v zasedací místnosti společnosti. Občas se setkání zúčastnily i jiné osoby, ale tyto osoby zde byly pouze pozorovatelé a do vývoje nemohli nijak zasahovat. Délka každodenního setkání trvala vždy od 10 do 15 minut. Během této schůzky každý člen týmu objasnil, na čem předešlý den pracoval a co má v plánu na den dnešní. Občas byly zmíněny překážky, které stály v cestě k dalšímu vývoji. Tyto překážky byly během Daily Scrum odstraněny nebo eliminovány tím, že ze vzniklé debaty o problému bylo vybráno jedno nejvhodnější a nejrychlejší řešení. Velká výhoda tohoto týmu byla komunikativnost a otevřená spolupráce. Tím se vzniklé problémy včas odhalily a eliminovaly.

Denní schůzky byly pro tým velmi prospěšné. Jelikož nemají společnou kancelář, byla omezena vzájemná komunikace na minimum. Právě díky Daily Scrum se vyřešili případné nesrovnalosti.

Na konci každé sprint se konala schůzka Sprint Review Meeting, která měla za úkol zhodnotit a shrnout toho, co se povedlo a co se nepovedlo. Této schůzky se v našem případě pravidelně zúčastnil pouze Product Owner, nikoliv zákazník, jak je doporučováno. Během této schůzky byla občas odhalena problematická místa a ta byla v následujícím sprintu odstraněna. Nedokončené úkoly byly vráceny zpět do Product Backlogu a opět se zhodnotila jejich náročnost a priorita.

Pro kontrolu vývoje produktu sloužil **Burn Down Chart**. Jednalo se o graf, díky kterému se sledoval průběh jednotlivých sprintů. Ten zobrazoval počet nesplněných úkolů v čase. Graf měl vždy klesající lineární průběh, což vypovídá o hladkém vývoji produktu. Graf byl veden v sešitu MS Excel, kde se automaticky aktualizoval. Burn Down Chart byl sice časově i finančně náročnější, ale v pozdější fázi vývoje se osvědčil svou přesností.

Na konci každého sprintu bylo zákazníkovi předáno demo – ukázka projektu, na kterou většinou zákazník reagoval připomínkami a novými požadavky. Tyto připomínky zákazník vložil do seznamu pořadí požadavků dle priorit. Díky připomínkám a novým návrhům k demoverzi se konečná doba projektu posunula o dva týdny. S tímto případem se však počítalo již při stanovování celkové doby trvání a tak to průběh nijak nenarušilo.



Obrázek 11 – Scrum process [15]

5.4 Vyhodnocení metodiky SCRUM

Zda byl projekt úspěšný a v jaké výši se odvíjelo srovnáním dané metodiky s jednotlivými dvanácti agilními principy.

Princip 1 Největší priorita je uspokojit zákazníka byl splněn. Jednotlivé fáze vývoje byly co nejkratší a časově adaptivní. Zákazník byl s jednotlivými demo verzemi spokojen a kladl jen připomínky nižších priorit.

Princip 2 Jsou vítané změny a připomínky zákazníka i v pozdních fázích projektu – tento princip byl splněn. I v pozdních fázích projektu zvládl tým bez menších problémů reagovat na připomínky a změny ze strany zákazníka. Bylo důležité, všechny tyto požadavky na změny a určení priorit dokumentovat.

Princip 3 Preferování kratšího časového období k dodání funkčního software – princip byl splněn. Jednotlivé cykly byly co nejkratší a časově adaptivní. V průběhu vývoje byly zákazníkovi předváděny demo verze, na které pak mohl reagovat a posoudit efektivnost projektu.

Princi 4 Důraz na častou spolupráci zákazníka s vývojáři. Podmínka byla splněn i v tomto principu díky pravidelným denním schůzkám Daily Scrum.

Princip 5 Podpora jednotlivců při budování projektu

Princip 6 Nejefektivnější a nejúčinnější metodou přesunu informací k týmu a uvnitř něj je osobní komunikace. Tento princip byl splněn díky schůzkám a také díky komunikativnosti jednotlivých členů týmu. Během celého projektu se nevyskytl problém s komunikací jak uvnitř týmu, tak se zákazníkem.

Princip 7 Hlavním měřítkem pokroku je fungující software. Tým měl k dispozici perfektně fungující software, což bylo hlavním důvodem k úspěšnému zvládnutí vývoje produktu.

Princip 8 Podpora trvale udržitelného rozvoje. Sponzoři, vývojáři a zákazníci udrželi stejnou rychlost po celou dobu vývoje.

Princip 9 Důraz na technické vybavení. Tým měl k dispozici jak perfektně fungující software, tak technicky dokonalý systém. U tohoto principu byla podmínka stejně jak u principu 7 splněna.

Princip 10 Jednoduchost. Cílem bylo zjednodušení práce na práci, kterou je nutno udělat.

Princip 11 Samostatná organizace týmu. Týmová organizace byla u vývoje produktu na velmi dobré úrovni. Na tým nebyla převedena velká odpovědnost a plánování.

Princip 12 Individuality a komunikace mají přednost před procesy a nástroji. Metodika zbytečně nezatěžovala zákazníka, nevyžadovala od něj specifické znalosti. V tom všem si tým vystačil sám.

Všechny agilní principy se podařilo splnit, a zavedení metodiky SCRUM na projekt ESS portál považuji za úspěšné.

5.5 Dokumentace projektu

Kvalitním a efektivním předávání informací mezi zúčastněnými osobami je důležité mít potřebnou dokumentaci, místo jejich uložení a nastavení práv a postupů s jejich disponováním.

Mezi takové dokumenty patřily:

Smlouva

Před-implementační analýza

Produktový backlog

Změnový dokument

Testovací scénáře

Předávací protokol

6 Doporučení metodik pro modelové případy

Obsahem této kapitoly bude aplikace tří vybraných metodik na tři vybrané modelové případy. Modelový případ je podrobně popsán v následujících podkapitolách. Cílem této kapitole bude doporučení metodiky pro jednotlivé případové studie, které byly následně aplikovány v praxi.

6.1 Aplikace metodiky Extrémní programování na případové studii

Cílem této kapitole je doporučit metodiku pro modelový případ. Metodiku Extrémní programování jsem aplikoval na vývoj systému Rezervace zasedacích místností ve společnosti Siemens.

6.1.1 Charakteristika případové studie

S rozvojem společnosti Siemens s.r.o. v Ostravě a s přibýváním nových zaměstnanců začaly vznikat problémy s jednotlivými zasedacími místnostmi. Stávající budova má 10 zasedacích místností, které slouží jak k přijímacím pohovorům, teamových meetingům tak hlavně k výuce cizích jazyků, na které klade společnost velký význam. S růstem nových zaměstnanců začaly stoupat nároky na využívání těchto prostor. Firma Siemens proto vyhodnotila vytvoření rezervačního systému těchto prostor za nezbytný. Náplní projektu měl být systém přehledu všech zasedacích místností, jejich vytížení a jejich obsazenosti. Jelikož byla situace se zasedacími místnosti neúnosná, chtěl zákazník zprovoznění systému v co nejkratší době, což bylo podle jeho představ do 6 týdnů od zadání.

Od dodaného softwaru očekával zákazník přehledný systém jednoduchý na ovládání, který přinese úplný přehled o všech zasedacích místnostech, jejich obsazenosti a časového vytížení. Dále požadoval, aby se rezervace daly vytvořit na delší časové období a pokud by to byla rezervace, která se opakuje vždy ve stejnou dobu, mělo být zajištěno nastavení její periodicity bez toho, aby se rezervace musela opakovaně nastavovat. Dalším požadavkem bylo, aby bylo z přehledu zřejmé, pro jaké účely byla místnost zarezervována.

6.1.2 Extrémní programování

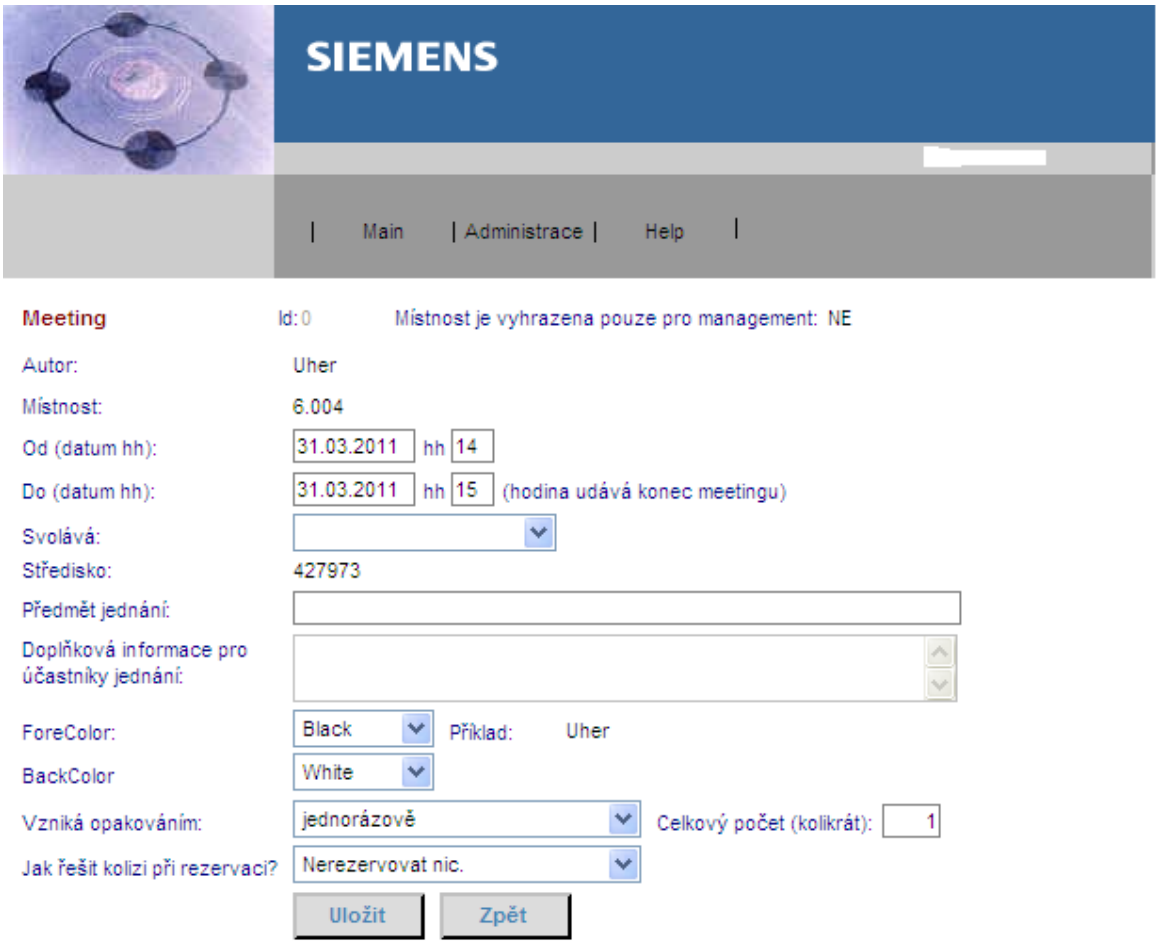
Průzkum

U extrémního programování je nezbytné naslouchání potřebám a požadavkům zákazníka. Na počátku celého projektu byly od zákazníka sepsány uživatelské příběhy zvané „User stories“, ve kterých byly zachyceny všechny požadavky zákazníka. Rozhovorem se zákazníkem a týmem byly tyto příběhy doplněny a vznikly „karty zadání“, ve kterém zákazník určil prioritu. Ostatní položky byly vyplněny až v následných etapách. Byl tu zadán požadavek na rezervaci zasedacích místností. Zaměstnanec si vybere jednu zasedací místnost na určitý čas a zjistí,

že je právě tato místnost již rezervovaná. Systém by mu měl nabídnout jinou, dosud nezarezervovanou místnost. V opačném případě po úspěšném rezervování místnosti a informuje ho o jeho rezervaci. Na tuto etapu měl tým pouze dva týdny. Byl zde předpoklad, že je zde zkušený tým vývojářů, kteří disponují zkušenostmi s požadovanou technologií. Na fázi průzkum pak navazovala fáze plánování.

Plánování

Plánování bylo realizováno pomocí „plánovací hry a iterační plánovací hry“. Tým musel odhadnout časovou náročnost jednotlivých karet zadání a vedoucí týmu setřídil karty podle požadavků zákazníka a jejich priorit. Karty byly seřazeny také podle rizika nepřesnosti odhadu. Následně byla zvolena sada karet, které byly implementovány. Byly naplánovány dvě iterace o délce celkem dva týdny. Během první iterace byla vytvořena kostra systému. Tato činnost trvala týmu celý jeden týden. V následné druhé iteraci došlo na doplnění systému o karty zadání s nejvyšším přínosem pro zákazníka. Na začátku každé z iterací proběhla plánovací hra, díky kterým se převedly vybrané karty zadání pro danou iteraci na úkolové karty programátora. Následně si každý programátor vzal část těchto karet zadání a rozebral je na jednotlivé úkoly, které byly vepsány do úkolových karet. Časová náročnost jednotlivých karet se u nás pohybovala v rozmezí několika dnů. Již na počátku odhadl programátor časovou náročnost jednotlivých úkolových karet a tuto náročnost vynásobil zátěžovým faktorem. V mnohých případech se stalo, že byl programátor přetížen a předal část svých karet svým méně pracovně zatíženým kolegům. Při časovém odhadu přebral programátor odpovědnost za implementaci úkolové karty. Nestal se případ, že by byli zaneprázdněni všichni programátoři a nemusel se tím řešit problém navrácení se do plánovací hry a přehodnocení rozsahu celého projektu. Při plánování se počítalo s tím, že při implementaci jednoho odhadovaného úkolu nebude programátor rušen a vznikly tak odhady v ideálním čase. Díky zátěžovému faktoru byl přepočítán ideální čas na kalendářní, tedy na skutečnou délku. Při programování projektu zabral programátorům úkol dvakrát tolik času, jež odhadovali. Bylo to dáno tím, že se v průběhu programování zabývali i jinými činnostmi. V následující fázi byla verze předána zákazníkovi



Obrázek 12 – Ukázka rezervace zasedací místnosti

Aplikace a zprovozňování

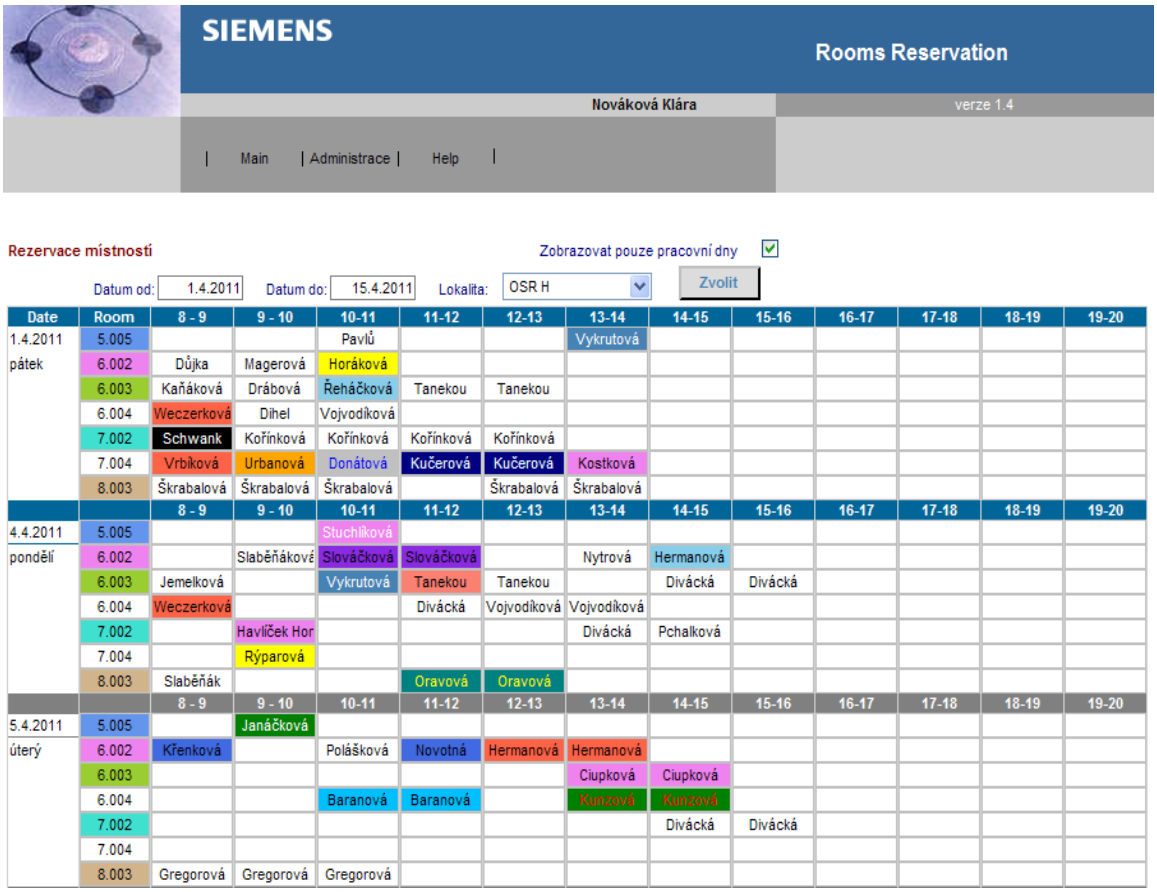
V této fázi byla zákazníkovi dodána a aplikována první verze. Důkladné testování je stěžejním stavebním kamenem metodiky. Bylo nasazeno první testování, které se týkalo problému, zda nebude již rezervovaná zasedací místnost poskytnuta k zarezervování jinému uživateli. Z důvodu, že akceptační testování probíhalo již v předchozích iteracích, se tato fáze ukázala jako úspěšná.

Údržba

Během údržby byl prováděn servis software. Byly zde také implementovány nové karty zadání.

Smrt

Smrt většinou nastává, pokud je projekt zcela hotový a zákazník k němu nemá žádné další připomínky a požadavky. Projekt byl sice dokončen, všechny požadavky byly splněny a všechny požadované karty zadání byly aplikovány. Dodavatel tohoto systému se však stal i servisem vyvinutého software a proto smrt v tomto případě nenastala.



Date	Room	8 - 9	9 - 10	10-11	11-12	12-13	13-14	14-15	15-16	16-17	17-18	18-19	19-20
1.4.2011	5.005			Pavlů			Vykrutová						
pátek	6.002	Důjka	Magerová	Horáková									
	6.003	Kaňáková	Drábová	Řeháčková	Tanekou	Tanekou							
	6.004	Weczerková	Dihel	Vojvodíková									
	7.002	Schwank	Kořínková	Kořínková	Kořínková	Kořínková							
	7.004	Vrbková	Urbanová	Donátová	Kučerová	Kučerová	Kostková						
	8.003	Škrabalová	Škrabalová	Škrabalová		Škrabalová	Škrabalová						
		8 - 9	9 - 10	10-11	11-12	12-13	13-14	14-15	15-16	16-17	17-18	18-19	19-20
4.4.2011	5.005			Stuchlíková									
pondělí	6.002		Slaběňáková	Slováčková	Slováčková		Nytrová	Hermanová					
	6.003	Jemelková		Vykrutová	Tanekou	Tanekou		Divácká	Divácká				
	6.004	Weczerková			Divácká	Vojvodíková	Vojvodíková						
	7.002		Havlíček Hor				Divácká	Pchalková					
	7.004		Ryparová										
	8.003	Slaběňák			Oravová	Oravová							
		8 - 9	9 - 10	10-11	11-12	12-13	13-14	14-15	15-16	16-17	17-18	18-19	19-20
5.4.2011	5.005		Janáčková										
úterý	6.002	Křenková		Polášková	Novotná	Hermanová	Hermanová						
	6.003						Ciupková	Ciupková					
	6.004			Baranová	Baranová		Kunzová	Kunzová					
	7.002							Divácká	Divácká				
	7.004												
	8.003	Gregorová	Gregorová	Gregorová									

Obrázek 13 – Ukázka rezervovaných zasedacích místností

6.2 Aplikace metodiky DSDM na případové studii

V této podkapitole jsem doporučil metodiku na systém Sledování nákladů jak oddělení, tak jednotlivých zaměstnanců společnosti.

6.2.1 Charakteristika případové studie

Díky systému Sledování nákladů má společnost přesný přehled o vynaložených nákladech oddělení a jednotlivých pracovníků. Jednotlivé oddělení (týmy) společnosti spadají pod určité nákladové středisko, na které se pak zaznamenávají všechny náklady, které toto oddělení vynaložilo na svou činnost. Pod nákladovým střediskem jsou zaznamenány všechny náklady spojené s činností tohoto oddělení. Jedná se o IT náklady, náklady na IT infrastrukturu, náklady na telefonní hovory, náklady na kancelářské potřeby, poštovné, pracovní pomůcky atd.

Od dodaného softwaru očekával zákazník přehledný systém jednoduchý na ovládání, který přinese úplný přehled o všech nákladech spojených jak s oddělením, tak jednotlivými zaměstnanci společnosti SIEMENS v požadovaném časovém období. Cílem tohoto software bylo transparentně zaznamenat všechny náklady a dopomoct tímto k úsporám společnosti v této finanční oblasti.

6.2.2 Dynamic System Development Method

Projekt musel projít všemi sedmi etapami životního cyklu. Z nichž první tři jsou sekvenční a další inkrementální a iterativní. Jednotlivé iterace mají předem stanovenou délku.

Úvod do projektu a studie proveditelnosti

Na počátku celého procesu vývoje bylo rozhodnuto, zda je tento projekt realizovatelný DSDM. Byly rozpoznány všechny hrozící rizika a také technické vybavení bylo označeno za dostačující. Na těchto fází vnikly dva projekty a to Zpráva o proveditelnosti a Rámcový plán vývoje. Tato etapa trvala vzhledem k šíři projektu 4 dny.

Obchodní studie

V této fázi došlo na zkoumání základní charakteristiky fungování firmy zákazníka a požadovaných technologií. Byl uspořádán workshop, kde se prošly jednotlivé aspekty zákazníka a vyvíjeného systému. Během workshopu určil zákazník priority jednotlivých úkolů. Při této etapě vznikl dokument Definice prostředí, kde jsou popsány prováděné procesy a zainteresovaní lidé ve firmě zákazníka. Druhým dokumentem, který při této etapě vznikl byla Definice architektury, kde je obsažen návrh architektury vyvíjeného systému a během etap se tento návrh měnil.

Stanovení modelu funkcí

Při této fázi vznikl kód prototypu, který je nazýván Funkční model. V této fázi, která je jak inkrementační tak iterační probíhala jak analýza, tak kódování.

Návrh a implementace

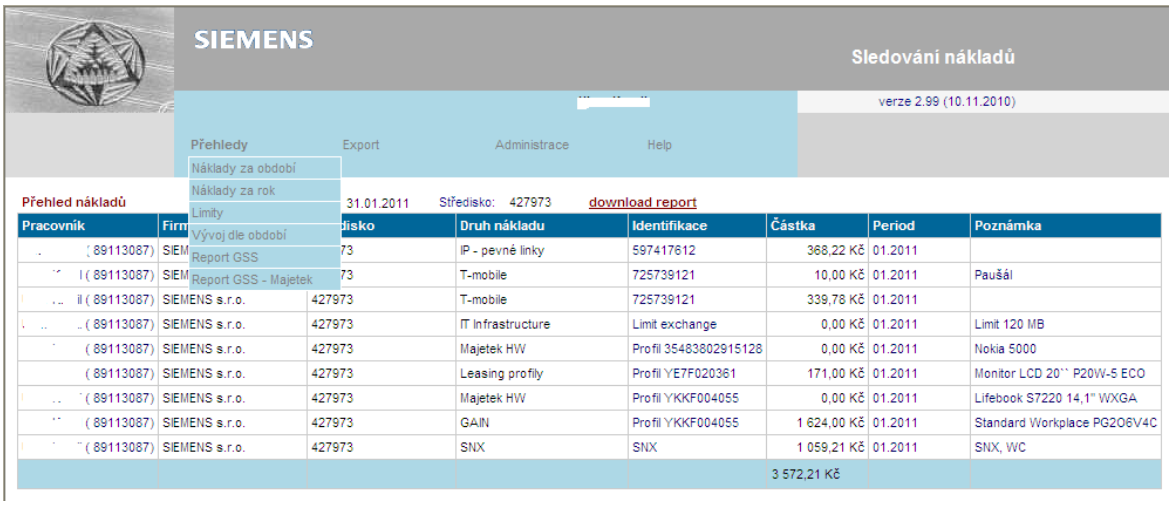
V průběhu této etapy se formoval vzhled systému a celý systém se začal sestavovat. Jako zpětná vazba sloužily připomínky zákazníka, který systém zkoušel během celé této etapy. Výsledkem této etapy byl systém, který svým otestováním splňoval základní požadavky a priority zákazníka.

Nasazení

Nasazení je jedna z nejhlavnějších fází vývoje požadovaného software. V této etapě se vyvíjený software nasazuje a implementuje u zákazníka. Tato fáze byla v závislosti složitosti systému rozdělena pouze na jednu iteraci. Zákazníkovi byl předán Uživatelský manuál a Zhodnocení projektu.

Závěr projektu

V této konečné fázi byl otestovaný projekt předán zákazníkovi. Projekt byl vyvinut úspěšně a zcela splňoval požadavky zákazníka.



Pracovník	Firm	disko	Druh nákladu	Identifikace	Částka	Period	Poznámka
(89113087)	SIEM	73	IP - pevné linky	597417612	368,22 Kč	01.2011	
l (89113087)	SIEM	73	T-mobile	725739121	10,00 Kč	01.2011	Paušál
il (89113087)	SIEMENS s.r.o.	427973	T-mobile	725739121	339,78 Kč	01.2011	
(89113087)	SIEMENS s.r.o.	427973	IT Infrastructure	Limit exchange	0,00 Kč	01.2011	Limit 120 MB
(89113087)	SIEMENS s.r.o.	427973	Majetek HW	Profil 35483802915128	0,00 Kč	01.2011	Nokia 5000
(89113087)	SIEMENS s.r.o.	427973	Leasing profily	Profil YE7F020361	171,00 Kč	01.2011	Monitor LCD 20`` P20W-5 ECO
(89113087)	SIEMENS s.r.o.	427973	Majetek HW	Profil YKKF004055	0,00 Kč	01.2011	Lifebook S7220 14,1" WXGA
(89113087)	SIEMENS s.r.o.	427973	GAIN	Profil YKKF004055	1 624,00 Kč	01.2011	Standard Workplace PG2O6V4C
(89113087)	SIEMENS s.r.o.	427973	SNX	SNX	1 059,21 Kč	01.2011	SNX, WC
					3 572,21 Kč		

Obrázek 14 – Ukázka Sledování nákladů na zaměstnance

6.3 Aplikace metodiky Cryslar Clear na případové studii

V třetí podkapitole jsem navrhl metodiku Crystal Clear na případovou studii vývoje docházkového systému ve společnosti.

6.3.1 Charakteristika případové studie

V třetí případové studii jsem navrhl metodiku Crystal Clear na zaznamenávání docházky zaměstnanců. Společnost zadala požadavek, na vytvoření softwarového systému na zaznamenávání pracovní doby zaměstnanců společnosti. Systém Docházka se řadí mezi nejdůležitější aplikace spojené s činností celé společnosti. Zákazník požadoval náročný, ale na ovládání jednoduchý systém na zaznamenávání docházky. Systém měl zaznamenávat jednotlivé příchody a odchody zaměstnanců, zobrazovat dny čerpané dovolené, účast na pracovní cestě a také dny, kdy čerpal zaměstnanec nemocenskou dovolenou. To vše se mělo prolínat s aplikací čerpání dovolené a s výpočtem mezd pracovníků. Cílem bylo vyprojektovat spolehlivý systém odměňování pracovníků za odpracovanou dobu.

6.3.2 Crystal Clear

Mapování a průzkum

Na počátku vývoje proběhlo mapování. Během mapování byl sestaven tým. Tým se skládal z šesti členů. V týmu byl zvolen hlavní vývojář, sponzor a uživatel. Průzkum sloužil k vyhodnocení stávající situace, a zda bude mít projekt smysl. Zkoumalo se, zda je schopný tým k realizaci požadavků zákazníka a zda bude mít výslední projekt pro samotného zákazníka smysl a bude efektivní. V této fázi byli identifikováni budoucí uživatelé vyvíjeného software, jako jsou zaměstnanci společnosti, dále byly určeny hlavní případy užití, jako je docházka a požadavky na systém. Během této etapy se také nastavily prvotní parametry metodiky, které byly v průběhu

workshopů upřesňovány. Tým se společně domluvil na rozsahu iterací a konvencí, které se při vývoji striktně dodržovaly. Probíhalo bleskové plánování, kdy byl tvořen plán na celý další měsíc. Doba jeden měsíc byla dána nutnost zavedení systému Docházka v co nejkratším termínu. Základním předpokladem fungujícího bleskového plánování bylo vypsání všech požadavků na plánovací karty, na kterých byly popsány časové náročnosti jednotlivých požadavků.

Bleskového plánování se zúčastnili všichni pracovníci podílející se na vývoji tohoto systému. Během 20 minutového Brainstormingu vznikly plánovací karty. Dále byly karty sestaveny dle nejdůležitějších priorit. Postupně byly jednotlivé karty kontrolovány a doplňovány o odhady vývojářů. Následoval výběr první verze běžící kostry. Během vývoje docházelo k identifikaci dalších verzí, optimalizaci podle projektových priorit až se postupně došlo k výslednému plánu. Délka celého mapování byla odhadnuta na tři pracovní dny.

Zdokonalující workshop

Vždy po uvolnění nové verze se sešli všichni členové týmu a konzultovali spolu názory a připomínky, vzniklé při vývoji. Jednotlivé nesrovnalosti, připomínky a problémy, které při vývoji software vznikly, se v následující fázi zdokonalily a eliminovaly. Občas zde byly navrženy nové postupy, které by se měli v dalších iteracích použít.

Dodávka

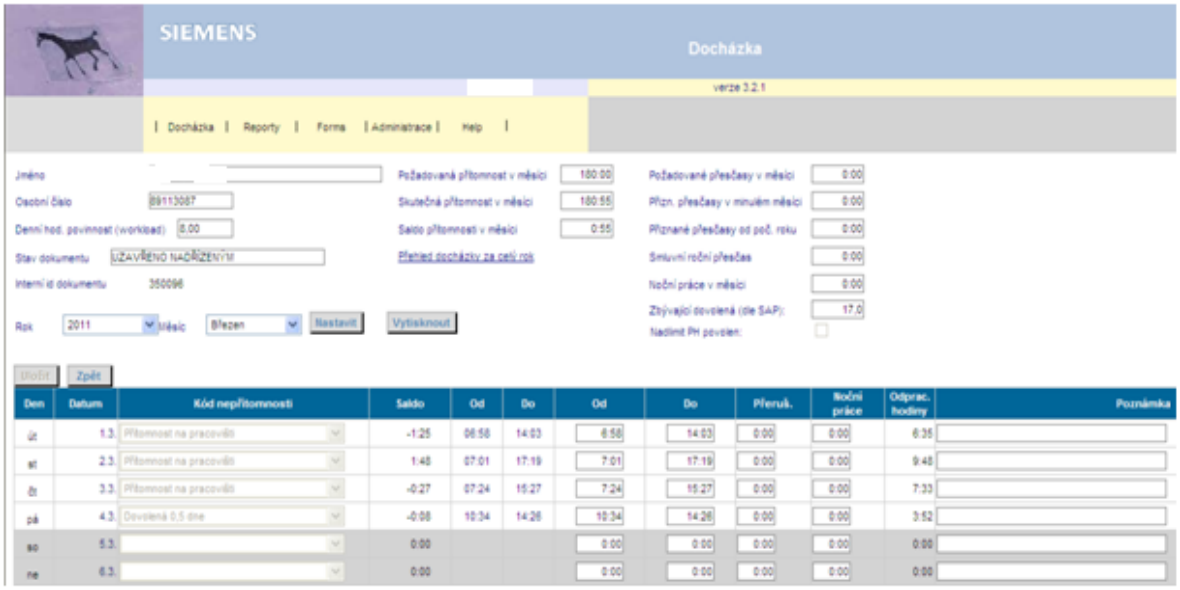
Po etapě mapování a průzkum následovala další časově náročná etapa nazvaná Dodávka. Tato etapa se skládala ze tří iterací, díky kterým se zrychlila zpětná vazba zákazníka na systém, a snížilo se riziko nespokojenosti zákazníka s vyvíjeným software. Jednalo se o v celku krátké iterace, takže bylo vyžadováno rozdělení požadavků na velmi malé úkony realizované v krátké době. Iterace v sobě obsahovala plánování, jednotlivé pracovní dny, iterační cykly, ukončení a oslavu projektu. Délka iteračního cyklu byla z části závislá na schopnostech a členů týmu. Délku iteračního cyklu z části ovlivňuje i technické vybavení celého týmu. V našem případě se jednalo o vyhovující technické zázemí a iterační cykly měli délku jednoho týdne. Výhodou bylo eliminace rizika problémů. Délka dodávky byla stanovena na jeden měsíc, kdy na konci iterace byla zákazníkovi dodána běžící kostra. Tým vývojářů prokazoval výborné znalosti a dovednosti z této oblasti, takže nedošlo k pracovnímu přetížení žádného z jeho členů.

Na začátku každého pracovního dne se uskutečnila denní schůzka všech členů vývojového týmu, kde se postupně projednávala činnost minulého dne a plánovala se činnost na den nadcházející. Každý z členů se k problematice vývoje vyjádřil, shrnul svoji práci z minulého dne a vyjádřil své názory a připomínky z vývoje. Pokud při vývoji nastal nějaký problém, byl tento problém projednáván právě na denní schůzce a tím se eliminovaly jednotlivé překážky, které bránily vývoji software. Denní schůzka v tomto případě nepřesáhla doporučenou dobu

15 minut. 15 minut každý den byla dostačující doba k projednání konkrétních názorů a připomínek k dané problematice.

Předání

Poslední fáze vývoje se nazývá Dodávka. Po měsíci byl konečný software hotový a implementovaný u zákazníka. Byla předána i potřebná dokumentace pro správné fungování daného systému.



Obrázek 15 – Ukázka aplikace Docházka

7 Závěr

Se zvyšováním dostupnosti výpočetní techniky se zvyšovala také dostupnost softwarových řešení. Projektů v informačních technologiích je stále více a jsou dostupné širšímu spektru lidí a také začaly růst požadavky na vývoj nových softwarových procesů. Na počátku všech metodik pro implementaci softwarových procesů stál „Vodopádový model“. Postupným zdokonalováním se přes „Spirálový model“ dostalo až k nejnovějším, v dnešní době stále rozrůstajícím se Agilním metodikám.

Prvním cílem této diplomové práce bylo získání přehledu o jednotlivých metodikách vývoje software a jejich porovnání, kterému byly věnovány první tři teoretické kapitoly diplomové práce. V první kapitole jsem popsal nejdostupnější a nejpoužívanější používané metodiky, v navazující druhé kapitole jsem tyto metodiky rozdělil podle charakteristických společných znaků na metodiky rigorózní a metodiky agilní. Ve třetí teoretické kapitole jsem jednotlivé vyjmenované metodiky navzájem porovnal podle jednotlivých principů, rolí a použitých postupů. Tímto bych označil první cíl za splněný.

Druhým cílem bylo vyhodnocení jednotlivých metodik a použití vhodných metodik na vybrané konkrétní projekty a modelové případy ve společnosti Siemens.

Řešení druhého cíle jsou věnovány poslední dvě praktické části diplomové práce. Ve čtvrté kapitole jsem navrhl použití metodiky SCRUM na ESS portál ve společnosti Siemens a popsal celou aplikaci této metodiky na tento projekt. Postupy vymezené metodikou vedly ke značnému snížení rizika neúspěchu díky častým dodávkám funkčních verzí. V poslední, páté kapitole jsem doporučil metodiky pro jednotlivé modelové případy. První z doporučených metodik byla metodika Extrémní programování, která byla aplikována na modelový případ Rezervace zasedacích místností ve společnosti. Druhou navrhovanou metodikou byla metodika DSDM, která měla být navržena na systém Sledování nákladů. Třetí, poslední doporučovanou metodikou byla metodika Crystal Clear, která byla navržena na systém Docházka.

Během vývoje systémů se navržené metodiky ukázaly jako úspěšné a schopné k aplikaci na dané případové studie. Ale i nejhodnější vybraná metodika neznamená záruku k úspěchu. Je nezbytné brát v úvahu lidský faktor a s ním spojená různá selhání. Na druhou stranu existují faktory, které lze bez problémů ošetřit a řídit jako jsou například komunikace, reálný odhad složitosti systému, důsledné testování a další.

Jednotlivé metodiky, navržené pro modelové případy byly společností aplikovány. Za nepodstatnější úspěch považuji spokojenost zákazníka. Společnost tyto navržené systémy aplikovala a s fungování systémů je spokojena. Systémy jsou nadále průběžně zdokonalovány

a rozvíjeny. Na základě tohoto konstatování můžu tvrdit, že byl i druhý cíl diplomové práce splněn.

Literatura

Knihy

[1] KADLEC, V. : *Agilní programování Metodiky efektivního vývoje softwaru*,Computer Press, 2004. ISBN 80-251-0342-0.

[2] Barry W. Boehm, :"*A Spiral Model of Software Development and Enhancement*," Ieee Computer Society Press, 2007. ISBN 047014873X.

[3] KRUCHTEN, Philippe.: *The Rational Unified Process An Introduction*. 2nd edition. Boston: Addison-Wesley, 2001. 298 s. ISBN 0-201-70710-1.

[4] SCHWABER, K. and M. Beedle, *Agile Software Development with Scrum*, Prentice-Hall, 2001. ISBN 0130676349

[5] Beck, K.:*Extreme Programming Explained*, Addison-Wesley Professional, 2000. 224 s. ISBN 0201616416.

[6] Schmuller, J.: *Myslíme v jazyku UML*, Grada, Praha 2001, ISBN 80-247-0029-8,

[7] Cockburn, A.: *Crystal Clear*: A Human-Powered Methodology for Small Teams, Addison-Wesley, 2005. 0-201-69947-8.

Dokumenty

[8] Vondrák, I.: *Úvod do softwarového inženýrství*.. 2002. Dec,

[9] Sommerville, I.: *Software Engineering (8th edition),* 2007.

[10] Royce, W.: *Managing the Development of Large Software Systems*, Proceedings of IEEE-WESCON, 1970.
zdroj http://www.cs.umd.edu/class/spring2003/cmsc838p/Process/waterfall.pdf

[11] Beck, K. a kolektiv: *Manifesto for Agile Software Development*,
zdroj: http://agilemanifesto.org/

[12] Metody tvorby informačních systémů
http://etext.czu.cz/img/skripta/64/pef_226-1.pdf

[13] Agilní Softwarové Procesy
zdroj: www.cs.vsb.cz/stolfa/vyuka/2007-2008/.../SAN%20-%20SCRUM+XP.ppt

Internet

[14] Vodopádový model - Wikipedie, otevřená encyklopedie,
zdroj: http://cs.wikipedia.org/wiki/Vodop%C3%A1dov%C3%BD_model

[15] Spirálový model - Wikipedie, otevřená encyklopedie,
zdroj: http://wiki.matfyz.cz/wiki/SWI026_Pavelka

[16] IBM Rational Unified Process - Wikipedie, otevřená encyklopedie,
zdroj: http://en.wikipedia.org/wiki/IBM_Rational_Unified_Process

[17] SCRUM - Wikipedie, otevřená encyklopedie,
zdroj: http://en.wikipedia.org/wiki/Scrum_%28development%29

[18] Moderní trendy v softwarovém inženýrství denní trendy,
zdroj: http://portal.zcu.cz/wps/PA_Courseware/DownloadDokumentu?id=40981

[19] Čím mohou přispět nejznámější agilní metodiky.
zdroj: www.agris.cz/etc/textforwarder.php?iType=2&iId=152838